DB2 for OS/390
Version 5

Administration Guide

SC26-8957-03

Note!

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

First Edition (June 1997)

This edition applies to Version 5 of IBM DATABASE 2 Server for OS/390 (DB2 for OS/390), 5655-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

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

This softcopy version is based on the printed edition of the book and includes the changes indicated in the printed version by vertical bars. Additional changes made to this softcopy version of the manual since the hardcopy manual was published are indicated by the hash (#) symbol in the left-hand margin.

Contents

Notices

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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

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

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

Programming Interface Information

This book is intended to help you to plan for and administer IBM DATABASE 2 Server for OS/390 (DB2 for OS/390).

This book also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by IBM DATABASE 2 Server for OS/390.

General-use Programming Interfaces allow the customer to write programs that obtain the services of DB2 for OS/390.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, 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 the IBM Corporation in the United States and/or other countries:

AD/Cycle
AIX
APL2
BookManager
C/370
CICS
CICS/ESA
CICS/MVS
COBOL/370
DATABASE 2
DataHub
DataRefresher
DB2
DFSMS
DFSMS/MVS
DFSMSdfp
DFSMSdss
DFSMShsm
DFSORT
Distributed Relational Database Architecture
DProp
DRDA
DXT
Enterprise System/9000
ES/9000
Hiperspace
IBM
IMS/ESA
Language Environment
MVS/DFP
MVS/ESA
MVS/SP
MVS/XA
NetView
OS/2
OS/390
Parallel Sysplex
QMF
RACF
RAMAC
Resource Measurement Facility
RETAIN
RMF
SAA
Systems Application Architecture
System/370
System/390
SQL/DS
VM/XA
VTAM

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

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

Section 1. Introduction

Chapter 1-1. Introduction to This Book and the DB2 Library

This chapter contains specific information about this book and a general overview of the DB2 library.

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.

How This Book Is Organized

This book consists of five sections, seven appendixes, and an index.

- Section 1. Introduction

  This section introduces the DB2 system and explains the concepts that relate to system and database administration.

- Section 2. Designing a Database

  This section describes the logical steps of database design and how to implement that design. It also shows you how to define referential constraints, load data into DB2 tables, and alter your database design.

- Section 3. Security and Auditing

  This section describes anything involved with the control of access to the DB2 subsystem, its data, or its resources. It also answers the question: How can I tell who has actually accessed the data?

- Section 4. Operation and Recovery

  This section covers starting and stopping DB2, entering commands, submitting work to be processed, and receiving messages. It also provides information on termination and restart, backing up and recovering databases, and recovery scenarios.

- Section 5. Performance Monitoring and Tuning

This section describes how to monitor the performance of your DB2 subsystem, and covers methods of improving DB2's speed and efficiency.

- Appendix A. DB2 Sample Tables
- Appendix B. Writing Exit Routines
- Appendix C. Reading Log Records
- Appendix D. Interpreting DB2 Trace Output
- Appendix E. Programming for the Instrumentation Facility Interface
- Appendix F. Sharing Read-Only Data
- Appendix G. Using Tools to Monitor Performance

How to Read the Syntax Diagrams

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

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

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

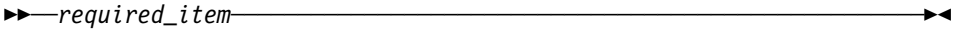
  The —▶ symbol indicates that the statement syntax is continued on the next line.

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

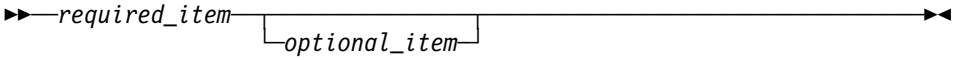
  The —▶◀ symbol indicates the end of a statement.

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

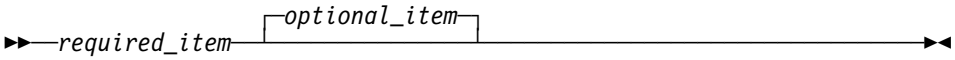
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

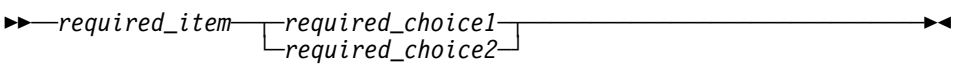


  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

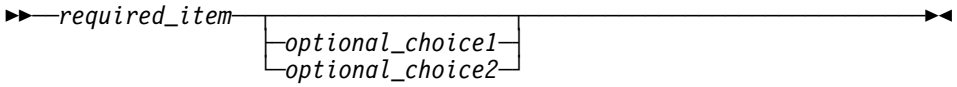


- If you can choose from two or more items, they appear vertically, in a stack.

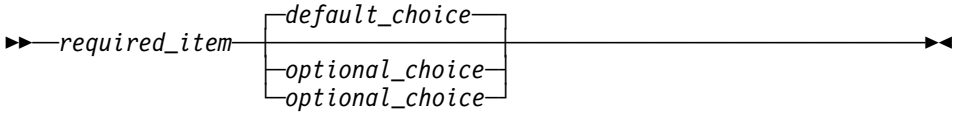
  If you *must* choose one of the items, one item of the stack appears on the main path.



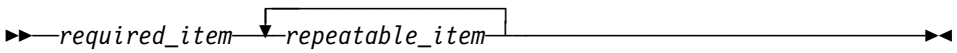
  If choosing one of the items is optional, the entire stack appears below the main path.



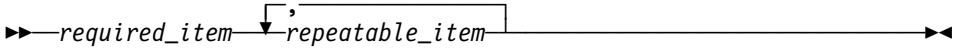
If one of the items is the default, it appears above the main path and the remaining choices are shown below.



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



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



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

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

How to Use the DB2 Library

Titles of books in the library begin with DB2 for OS/390 Version 5. However, references from one book in the library to another are shortened and do not include the product name, version, and release. Instead, they point directly to the section that holds the information. For a complete list of books in the library, and the sections in each book, see the bibliography at the back of this book.

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

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task called *end use*. Other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. The tasks associated with DB2 are grouped into the following major categories (but supplemental information relating to all of the below tasks for new releases of DB2 can be found in *Release Guide*):

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

If you will be using data sharing then you also need *Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

End use: End users issue SQL statements to retrieve data. They can also insert, update, or delete data, with SQL statements. They might need an introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. This information is found in *Application Programming and SQL Guide* and *SQL Reference*.

End users can also issue SQL statements through the Query Management Facility (QMF) or some other program, and the library for that program might provide all the instruction or reference material they need. For a list of some of the titles in the QMF library, see the bibliography at the end of this book.

Application Programming: Some users access DB2 without knowing it, using programs that contain SQL statements. DB2 application programmers write those programs. Because they write SQL statements, they need *Application Programming and SQL Guide*, *SQL Reference*, and *Call Level Interface Guide and Reference* just as end users do.

Application programmers also need instructions on many other topics:

- How to transfer data between DB2 and a host program—written in COBOL, C, or FORTRAN, for example
- How to prepare to compile a program that embeds SQL statements
- How to process data from two systems simultaneously, say DB2 and IMS or DB2 and CICS
- How to write distributed applications across platforms
- How to write applications that use DB2 Call Level Interface to access DB2 servers
- How to write applications that use Open Database Connectivity (ODBC) to access DB2 servers
- How to write applications in the Java programming language to access DB2 servers

The material needed for writing a host program containing SQL is in *Application Programming and SQL Guide* and *Application Programming Guide and Reference for Java*. The material needed for writing applications that use DB2 Call Level Interface or ODBC to access DB2 servers is in *Call Level Interface Guide and Reference*.

For handling errors, see *Messages and Codes*.

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

System and Database Administration: *Administration* covers almost everything else.

If you are involved with DB2 only to design the database, or plan operational procedures, you need *Administration Guide*. If you also want to carry out your own

plans by creating DB2 objects, granting privileges, running utility jobs, and so on, then you also need:

- *SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges
- *Utility Guide and Reference*, which explains how to run utilities
- *Command Reference*, which explains how to run commands

If you will be using data sharing, then you need *Data Sharing: Planning and Administration*, which describes how to plan for and implement data sharing.

Additional information about system and database administration can be found in *Messages and Codes*, which lists messages and codes issued by DB2, with explanations and suggested responses.

Diagnosis: Diagnosticians detect and describe errors in the DB2 program. They might also recommend or apply a remedy. The documentation for this task is in *Diagnosis Guide and Reference* and *Messages and Codes*.

How to Obtain DB2 Information

DB2 on the Web

Stay current with the latest information about DB2. View the DB2 home page on the World Wide Web. News items keep you informed about the latest enhancements to the product. Product announcements, press releases, fact sheets, and technical articles help you plan your database management strategy. Technical professionals can access DB2 publications on the Web and follow links to other Web sites with more information about DB2 family and OS/390 solutions. Access DB2 on the Web with the following URL:

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

DB2 Publications

The DB2 publications are available in both hardcopy and softcopy format. Using online books on CD-ROM, you can read, search across books, print portions of the text, and make notes in these BookManager books. With the appropriate BookManager READ product or IBM Library Readers, you can view these books on the MVS, VM, OS/2, DOS, AIX and Windows platforms.

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

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

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

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

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

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

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

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

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

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

See your IBM representative for assistance in ordering the collection.

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

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

Online Library Omnibus Edition OS/390 Collection, SK2T-6700, in English

IBM Online Library MVS Collection Kit, SK88-8002, in Japanese, for viewing on DOS and Windows platforms

How to Order the DB2 Library

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

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

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

DB2 Classes

IBM Education and Training offers a wide variety of classroom courses to help you quickly and efficiently gain DB2 expertise. Classes are scheduled in cities all over the world. For more information, including the current local schedule, please contact your IBM representative.

Classes can also be taught at your location, at a time that suits your needs. Courses can even be customized to meet your exact requirements. The diagrams below show the DB2 curriculum in the United States. *Enterprise Systems Training*

Solutions, GR28-5467 describes these courses. You can inquire about or enroll in them by calling 1-800-IBM-TEACh (1-800-426-8322).

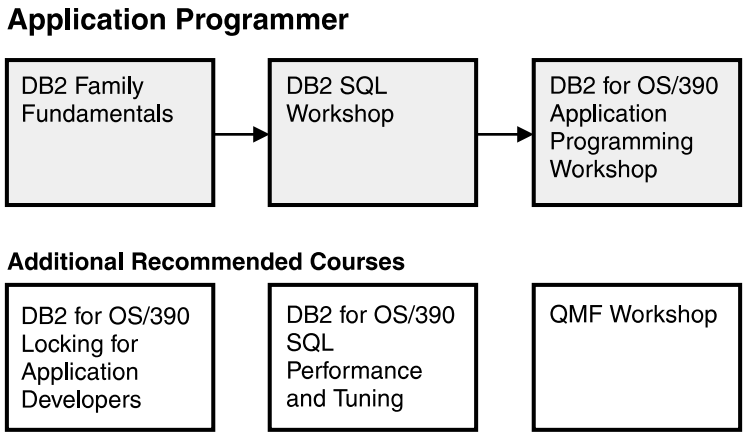


Figure 1. Application Programmer Curriculum

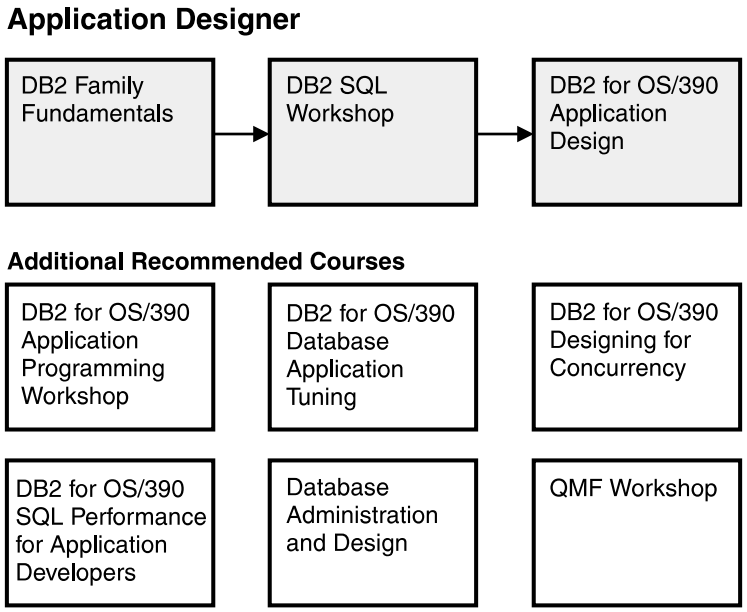


Figure 2. Application Designer Curriculum

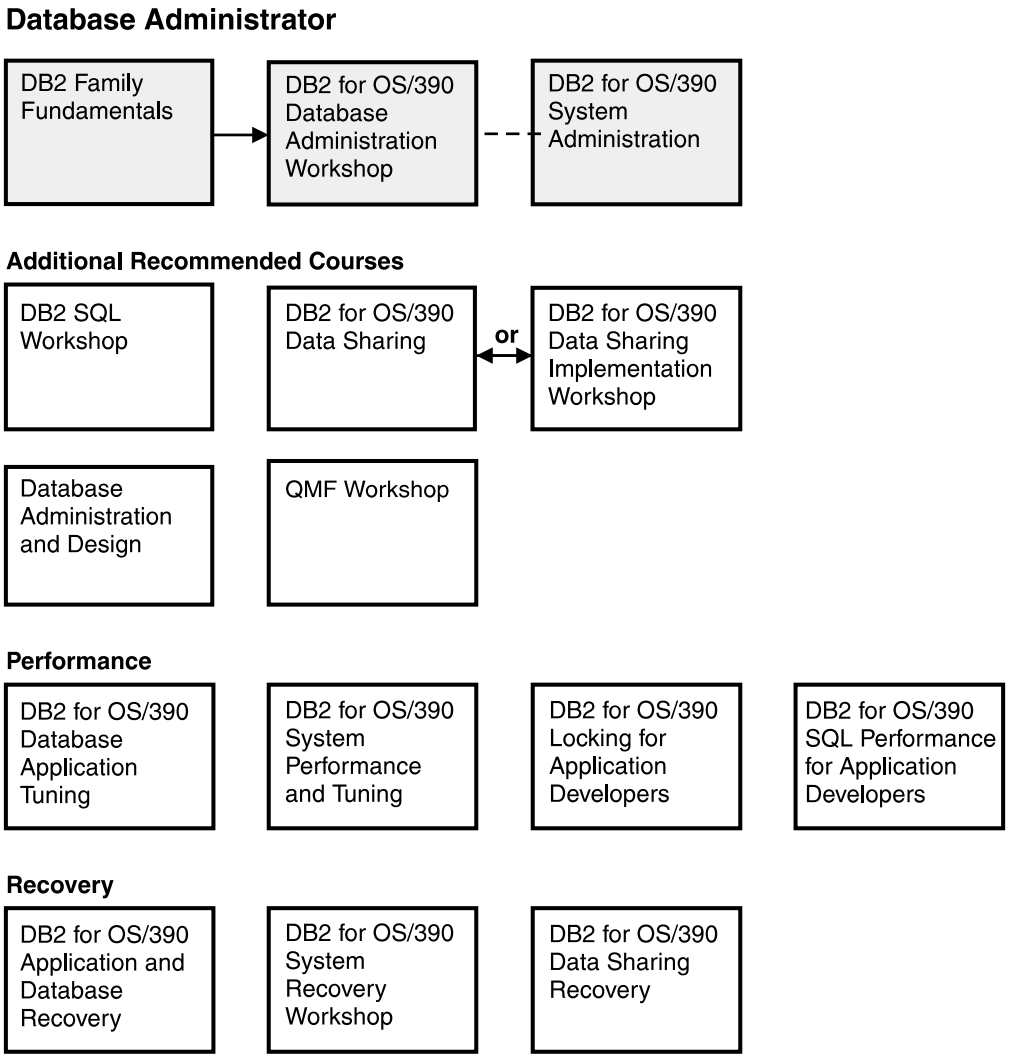


Figure 3. Database Administrator Curriculum

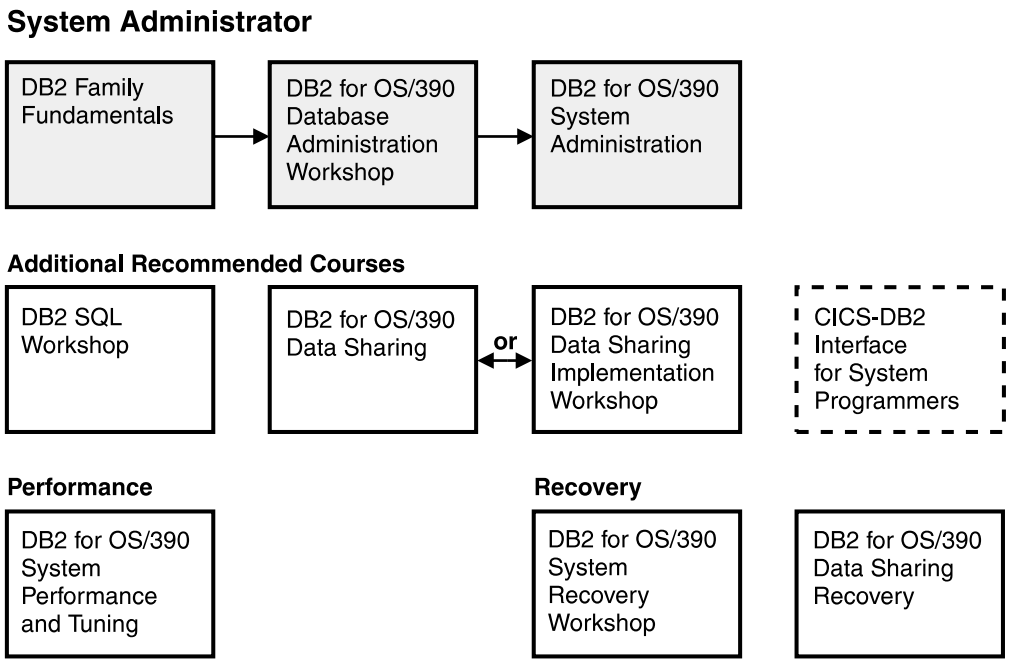


Figure 4. System Administrator Curriculum

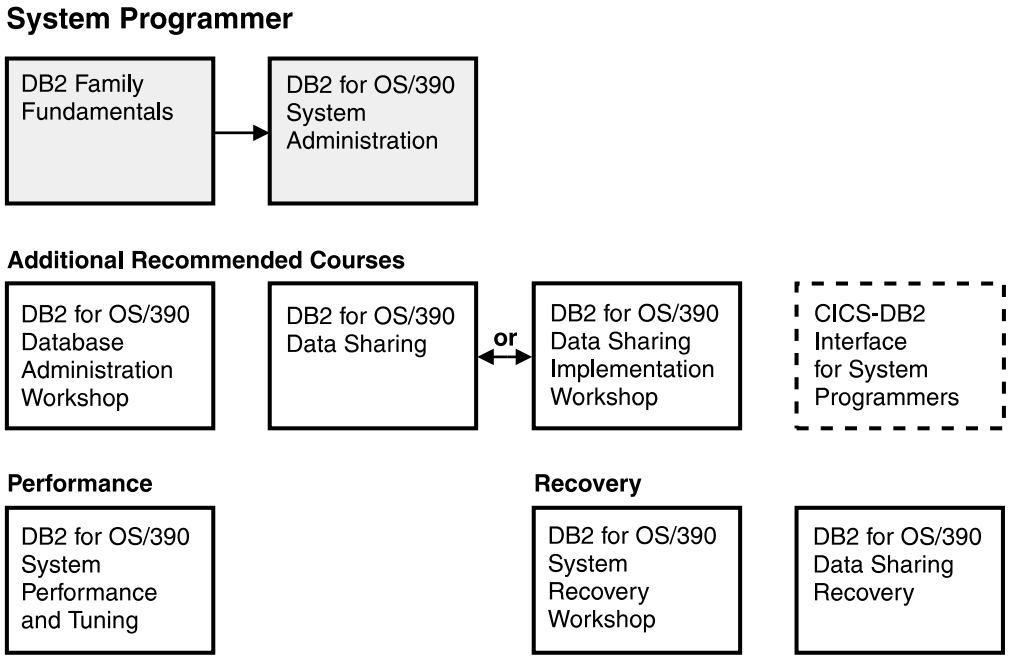


Figure 5. System Programmer Curriculum

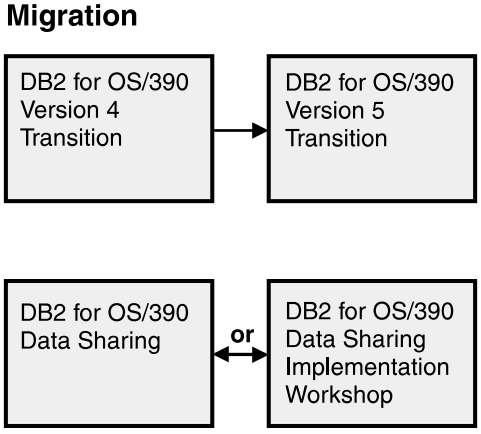


Figure 6. Migration Curriculum

Summary of Changes to DB2 for OS/390 Version 5

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

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

Server Solution

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

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

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

Net.Data for OS/390

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

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

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

DB2 Installer

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

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

DB2 Estimator for Windows

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

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

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

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

DB2 Visual Explain

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

Workstation-based Performance Analysis and Tuning

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

DATABASE 2 Performance Monitor (DB2 PM)

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

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

This feature can be ordered with DB2 Version 5.

Performance

Sysplex Query Parallelism

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

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

Prepared Statement Caching

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

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

Reoptimization

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

Faster Transactions and Batch

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

Faster Utilities

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

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

Other Performance Enhancements

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

Increased Capacity

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

Improved Availability

Online REORG

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

Data Sharing Enhancements

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

Tracker site for disaster recovery

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

Client/Server and Open Systems

Native TCP/IP Network Support

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

Stored Procedures

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

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

Dynamic Query and Network Performance

Improvements for DRDA Applications

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

Improved Application Portability

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

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

Improved Security

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

User Productivity

Improved SQL Compatibility

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

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

New Access Choice

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

Image Copy Enhancements

The COPY, LOAD, and REORG utilities provide:

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

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

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

Other Usability Enhancements

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

- Trace records for IFCID 0022 now include most information in the PLAN_TABLE.
- An increase from 127 to 255 rows on a page improves table space processing and eliminates the need for compression.
- Install SYSOPR can recover objects using the START DATABASE command.
- A filtering capability for DISPLAY BUFFERPOOL limits statistics information to a specified set of page sets.
- You can enter comments within the SYSIN input stream for DB2 utilities.

Summary of Changes to This Book

Section 2 of Administration Guide: has the following changes:

- Information about creating temporary tables.
- The term *parent key* refers to either a primary key or unique key. Foreign keys can reference primary and unique keys.
- Steps on how to alter a table from EBCDIC and ASCII.
- The section on "Consider Creating Views of Your Tables" on page 2-17 has been added at the end of Chapter 2-2.
- The section on "Implementing Referential Constraints" on page 2-104 has been moved from Chapter 2-3 to Chapter 2-9.

Section 3 of Administration Guide: has the following changes:

- "Chapter 3-2. Controlling Access to DB2 Objects" on page 3-13 includes information about caching authorization IDs for packages.
- "Chapter 3-4. Controlling Access to a DB2 Subsystem" on page 3-63 has extensive changes describing the following new and changed functions:
  - Setting up security for TCP/IP connections
  - Configuring the DB2 server for the Distributed Computing Environment (DCE) security
  - Using RACF PassTickets
  - Setting up security for stored procedures in WLM-established address spaces
  - Options for extended security

Section 4 of Administration Guide: has the following changes:

- Information is added about operation and recovery for applications that use Recoverable Resource Manager Services attachment facility (RRSAF).
- "Monitoring and Controlling Stored Procedures" on page 4-72 is expanded to include stored procedures in WLM-established address spaces.
- "Controlling Connections to Remote Systems" on page 4-62 is expanded to include monitoring distributed threads that use TCP/IP.

Section 5 of Administration Guide: has the following changes:

- For a list of performance enhancements in Version 5, see "Enhancements in DB2 Version 5" on page 5-17.

- Information about how temporary tables use work file space is included in "Work File Data Sets" on page 5-93.
- "MVS Performance Options for DB2" on page 5-108 is revised to include information about setting up dispatching priorities for stored procedures address spaces. There is also additional information about how I/O priority is determined when you are running in goal mode.
- "Using Workload Manager to Set Performance Objectives" on page 5-124 includes information about how to set performance objectives for stored procedures that run in a WLM-established address space.
- Information about the LOCKPART clause of CREATE and ALTER TABLESPACE is included in "Chapter 5-7. Improving Concurrency" on page 5-137. That chapter also includes information about the KEEP UPDATE LOCKS clause.
- "Chapter 5-8. Tuning Your Queries" on page 5-203 contains information about collecting statistics on correlated columns and about the new REOPT(VARS) bind option.
- "Chapter 5-9. Maintaining Statistics in the Catalog" on page 5-243 has information about when to reorganize table spaces and indexes. Some of this information was previously in an appendix.
- Guidance about how to partition for the best parallel performance has been added to "Chapter 5-10. Using EXPLAIN to Improve SQL Performance" on page 5-261.
- "Chapter 5-11. Monitoring and Tuning in a Distributed Environment" on page 5-315 is reorganized to put stored procedures information in one place. That section also contains information about how to set up application environments for WLM-established stored procedures.

Appendixes of Administration Guide: has the following change:

- Some information in "Appendix G. Using Tools to Monitor Performance" on page X-173 has been deleted or moved to other places in the library.

Chapter 1-2. System Planning Concepts

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

- "The Relational Database" is a broad introduction to DB2.
- "The Structure of DB2" on page 1-22 describes the elements you deal with when using DB2.
- "Control and Maintenance of DB2" on page 1-33 briefly describes commands and utility jobs.
- "DB2 and the MVS Environment" on page 1-36 explains how DB2 operates with certain related IBM products.

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

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

The Relational Database

DB2 is a *relational database management system*. In a relational database, data is perceived to exist in one or more tables, each containing a specific number of *columns* and a number of unordered *rows*. Each column in a row is related in some way to the other columns. Thinking of the data as a collection of tables gives you an easy way to visualize the stored data and enables you to explain your needs in easy-to-understand terms. Table 1 shows the department table (DSN8510.DEPT) of the sample database. The table contains four columns (DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT) and nine rows.

Table 1. Example of a DB2 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

DB2 accesses data by referring to its content instead of its location or organization in storage. The rows of a relational table have no fixed order. The order of the columns, however, are always the order in which you specified them when defining the table.

A DB2 database involves more than just a collection of tables. It also includes table spaces, storage groups, views, indexes, and other items. These are all collectively referred to as DB2 *structures*.

Sample Tables: The examples in this book are based on the set of tables described in Appendix A of *Administration Guide*. Those 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.

Structured Query Language (SQL)

The language used to access the data in DB2 tables is called SQL. SQL contains both data definition statements and data manipulation statements. Data definition statements define the structures of the database. Data manipulation statements manipulate (retrieve, insert, delete, and update) data in tables. SQL also performs functions that are neither data definition nor data manipulation, such as granting or revoking authorization to use resources.

You can execute statements written in SQL in the following ways:

- Interactively, using the SQL processor using file input (SPUFI) or Query Management Facility (QMF), another IBM licensed program.
- # By embedding them in application programs written in Ada, APL2, assembler,
  # BASIC, C, C++, COBOL, FORTRAN, Java, PL/I, Prolog, or REXX.
- | By using a DB2 Call Level Interface application program. For more information
  | about using DB2 CLI, see Call Level Interface Guide and Reference.

An important aspect of SQL is that it is nonprocedural; that is, when you use SQL you specify *what* you want done, not *how* to do it. In particular, to access data you need only name the table and column where it can be found; you do not have to describe an access method.

Below is an example of an SQL SELECT statement. It selects (retrieves) the department number (DEPTNO) and department name (DEPTNAME) columns from the department table (DSN8510.DEPT). It retrieves only those rows where the administrating department (ADMRDEPT) is D01.

```
SELECT DEPTNO, DEPTNAME
  FROM DSN8510.DEPT
  WHERE ADMRDEPT = 'D01';
```

You can also use SQL statements to insert new rows into a table, update existing rows, or delete rows.

For an elementary introduction to SQL and detailed instructions for using SPUFI, see Section 2 of *Application Programming and SQL Guide*. For a complete reference to the SQL language, see *SQL Reference*.

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:

The brief descriptions here show how the structures fit into an overall view of DB2. "Chapter 2-9. Implementing Your Design" on page 2-79 contains detailed information about each structure and explains how to use SQL to define them. There are other structures, such as group buffer pools and shared communications areas, that relate specifically to the data sharing environment. See *Data Sharing: Planning and Administration* for more information.

Figure 7 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.

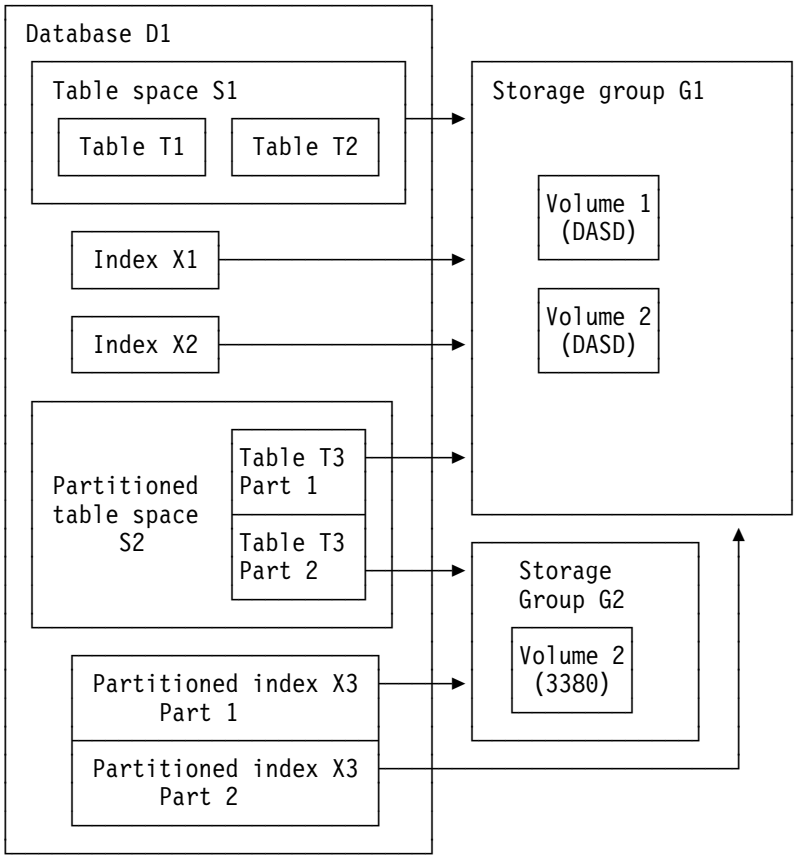


Figure 7. A Hierarchy of DB2 Structures

Databases

In DB2, a *database* is a set of DB2 structures. When you define a DB2 database, you give a name to an eventual collection of tables and associated indexes, as well as to the table spaces in which they reside. A single database, for example, 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.

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. The default database is predefined in the installation process; its default buffer pool is BP0, and its default DB2 storage group is SYSDEFLT.

Storage group SYSDEFLT is created when you install DB2. After that, 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 5, DB2 adopts the default database and default storage group you used in Version 4. You have the same authority for Version 5 as you did in Version 4.

DB2 Storage Groups

A DB2 *storage group* is a set of volumes on direct access storage devices (DASD). The volumes hold the data sets in which tables and indexes are actually stored. The description of a storage group names the group and identifies its volumes and the VSAM catalog that records the data sets.

All volumes of a given storage group must have the same device type. But, as Figure 7 on page 1-23 suggests, parts of a single database can be stored in different storage groups.

DFSMS storage groups are also discussed in this book. They are not the same as DB2 storage groups. When DFSMS storage groups are discussed, the reference will be explicit.

Table Spaces

A *table space* is one or more data sets in which one or more tables are stored. A table space can consist of a number of VSAM data sets , which can together contain up to 64 gigabytes of data for tablespaces not defined as large. A LARGE table space can consist of up to 254 data sets, or 1 terabyte of data. 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 DASD in one operation. Refer to the following sections for maximum sizes of each table space type. You should also refer to Appendix A of *SQL Reference* for specific limits.

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 control whether the table space is *partitioned*, *segmented*, or *simple*.

Partitioned Table Spaces: In a partitioned table space, the available space is divided into separate units of storage called *partitions*, each containing a part of one table. Although the partitions can be independently assigned to separate storage groups, the entire collection of data is logically a single table. A partition for a table space not defined as large can be 1, 2, or 4 gigabytes in length, depending on the number of partitions contained in the entire table space. If less than 16 partitions are defined on the table space, then each partition's maximum size is 4 gigabytes. For a table space that is defined as large, the maximum size of a partition is 4 gigabytes, for 1 to 254 partitions. See “Chapter 2-5. Designing Indexes” on page 2-51 for more information on partitioned table spaces.

Partitioning a table space provides several advantages for large tables:

- Improved data availability. You can perform normal maintenance on one partition of the table while the rest of the table remains available for utility or SQL processing.
- Improved utility performance. A utility can work on all partitions simultaneously instead of working on one partition at a time. Also, different utilities can work on different partitions simultaneously. This can significantly reduce the amount of time needed for a utility to finish.
- Improved query response time. When DB2 scans data to answer a query, it scans through partitions simultaneously instead of scanning through the entire table space from beginning to end. This improvement is most significant for queries that are complex or require DB2 to scan a lot of data.
- Improved efficiency in table space and index scans. DB2 can limit a scan to a subset of the partitions in both table space and index scans. Only the specific partitions that are needed are actually scanned.

Segmented Table Spaces: A segmented table space is intended to hold more than one table. The available space is divided into groups of pages called *segments*, each the same size. Each segment contains rows from only one table. To search all the rows for one table, it is not necessary to scan the entire table space, but only the segments that contain that table. If a table is dropped, its segments become immediately reusable. A segmented table space can have between 1 and 32 VSAM linear data sets, all of which are either user-defined or in the same storage group. The maximum size of a data set in the segmented table space is 2 gigabytes. And so, the maximum size of a segmented table space is 64 gigabytes (2 gigabytes multiplied by 32 data sets). See “Chapter 2-5. Designing Indexes” on page 2-51 for more information on segmented table spaces.

Simple Table Spaces: If a table space is not partitioned or segmented, it is called *simple*. A simple table space can contain more than one table, but the rows of different tables are not kept separate. To find all the rows of one table can require scanning the entire table space. If a table is dropped, its rows are not deleted. The space occupied by the rows does not become available until the table space is reorganized. All tables in a table space must be either user-defined or in the same storage group. The maximum size of a data set in the simple table space is 2 gigabytes. And so, the maximum size of a simple table space is 64 gigabytes (2 gigabytes multiplied by 32 data sets).

Tables

All data in a DB2 database is presented in *tables*—collections of rows all having the same columns. When you create a table in DB2, you define an ordered set of columns.

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00

A few rows of the sample department table are shown above. The ordered set of columns contains DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT. Each row contains data for a single department; the columns represent, respectively, its number, its name, the employee number of its manager, and the number of the department to which it reports. The whole table is a collection of rows, each containing those columns.

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.

The storage representation of a row is called a *record*, and the storage representation of a column is called a *field*. For most tasks discussed in this book, you do not need to know what a record looks like. For instances when you do need to know, see Appendix B of *Administration Guide*.

All the data in a given column must be of the same data type. For example, the data in the DEPTNAME column of the table is varying-length character (VARCHAR). DB2 data types are described in “Specifying Data Types” on page 2-44 .

A table can have a *primary key*. A primary key is a column or set of columns whose values uniquely identify each row. In the sample department table, DSN8510.DEPT, the DEPTNO (department ID) column is a primary key. Columns of other tables can be *foreign keys*, whose values must be equal to values of the primary key of the first table. In the sample employee table, DSN8510.EMP (see Appendix A of *Administration Guide*) the column that shows what department an employee works in is a foreign key; its values must be values of the department ID column in the department table. 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. Automatic enforcement of referential integrity is described in “Chapter 2-3. Maintaining Data Integrity” on page 2-19.

Indexes

An *index* is an ordered set of pointers to the data in a DB2 table. The index is stored separately from the table. 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 check, repair, or reorganize it.

You can create an index on a table any time after you create the table, and you can create the index either before or after you load data into the table. Except for changes in performance, users of the table are unaware that an index is being used. DB2 decides whether or not to use the index to access the table.

Each index occupies its own *index space*. A nonpartitioned index on a table space that **is not** defined as *large* can contain from 1 to 32 linear data sets, each data set capable of holding 2 GB of data . A nonpartitioned index on a table space that **is** defined as *large* can contain from 1 to 128 VSAM linear data sets, each capable of holding up to 4 GB of data, for a total of .5 terabytes. When you create an index, an index space is automatically defined in the same database as the table.

Refer to Appendix A of *SQL Reference* for specific limits.

You can use indexes to:

- Improve performance. In many cases, access to data is faster with an index than without.
- Ensure uniqueness. A table with a unique index cannot have two rows with the same values in the column or columns that form the index key.

Both of these topics are discussed further in “Chapter 2-5. Designing Indexes” on page 2-51 .

Partitioned Indexes: A partitioned index is created with the keyword PART, on a table in a partitioned table space that is divided into multiple index spaces.

A partitioned index consists of between 1 and 64 partitions for a table space that is not defined as large and between 1 and 254 partitions for a table space that is defined as large. Each of these partitions has a one to one correspondence to a VSAM data set. Each partitioned table space always has one table and one partitioned index defined on it. The maximum size of a partitioned index defined on a table space that is not defined as large is 1, 2 or 4 GB depending on the number of partitions. The maximum size of a partitioned data set defined on a table space that is defined as large is 4 GB. A partitioned index is always a clustering index.

Clustering Indexes: A clustering index determines the approximate order in which records of the base table are stored. Therefore, DB2 can access the entire table in the sequence of the clustering key faster than in any other sequence. Each table can have only one clustering index.

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. Views can also be based on other views or on a combination of views and tables. Figure 8 on page 1-28 shows some of the possible relationships between tables and the views that users see of them.

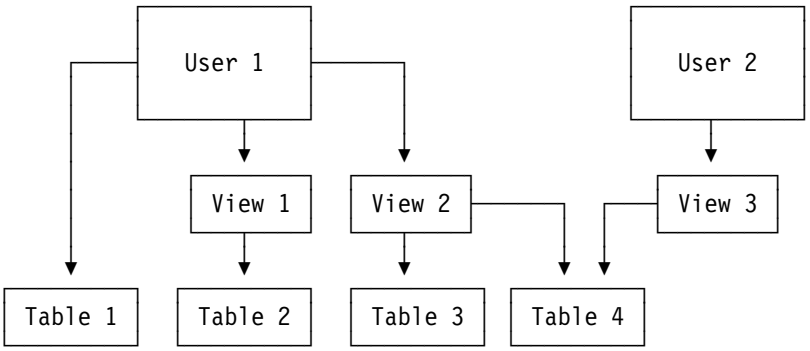


Figure 8. Relationship between Tables and Views

A view looks just like a table and can be used as though it were a table. You can use a view name in the FROM clause of an SQL SELECT statement, in the same way you use a table name. Although some operations cannot be performed on a view, often a user need not know that an apparent table is actually a view.

A table has a storage representation, but a view does not. When you define a view, DB2 stores the definition of the view in the DB2 catalog. But DB2 does not store any data for the view itself, because the data already exists in the base table or tables. Because no data is stored for the view, you cannot create an index on it. However, an index on a base table can improve the performance of operations on the view. For more information on views, see Section 2 of *Administration Guide*.

System Structures

DB2 system structures described in this section include:

DB2 Catalog

The DB2 catalog consists of tables of data about everything defined to the DB2 system. The DB2 catalog is contained in system database DSNDB06. 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. Appendix D of *SQL Reference* describes all the DB2 catalog tables.

| For Version 5, the communications database (CDB) is moved into the catalog.

| DB2 has extensive support to help move your applications into the next millennium.
| The Version 5 catalog supports timestamps generated both before and after the
| year 2000.

To illustrate the use of the catalog, here is a brief description of some of what happens when the employee table is created:

- To record the name of the structure, its owner, its creator, its type (alias, table, or view), the name of its table space, and the name of its database, DB2 inserts a row into the catalog table SYSIBM.SYSTABLES.
- To record the name of the table to which the column belongs, its length, its data type, and its sequence number in the table, DB2 inserts rows into SYSIBM.SYSCOLUMNS for each column of the table.
- To increase by one the number of tables in the table space DSN8S51E, DB2 updates the row in the catalog table SYSIBM.SYSTABLESPACE.
- To record that the owner (DSN8510) of the table has all privileges on the table, DB2 inserts a row into table SYSIBM.SYSTABAUTH.

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

For catalog data set naming conventions, see *Installation Guide*.

DB2 Directory

The DB2 directory contains information required to start DB2, and DB2 uses the directory during normal operation. You cannot access the directory using SQL. 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 following table spaces is contained in a VSAM linear data set:

- SCT02 is the skeleton cursor table space (SKCT).
- SPT01 is the skeleton package table space.
- SYSLGRNX is the log range table space.
- SYSUTILX is the system utilities table space.
- DBD01 is the database descriptor (DBD) table space.

The directory contains DSNSCT02, an index space for SCT02; DSNSPT01 and DSNSPT02, index spaces for SPT01; DSNLLX01 and DSNLLX02, indexes for SYSLGRNX; and DSNLUX01 and DSNLUX02, the indexes for SYSUTILX.

For directory data set naming conventions, see *Installation Guide*.

Skeleton Cursor Table: The skeleton cursor table space (SCT02) contains a table that describes the internal form of SQL statements of application programs. When you bind a plan, DB2 creates a skeleton cursor table in SCT02. The index space for the skeleton cursor table is DSNSCT02.

Skeleton Package Table: The skeleton package table space (SPT01) contains a table that describes the internal form of SQL statements in application programs. When you bind a package, DB2 creates a skeleton package table in SPT01. The index spaces for the skeleton package table are DSNSPT01 and DSNSPT02.

Log Range: DB2 inserts a row in the log range table space (SYSLGRNX) every time a table space or partition is opened and updated, and it updates SYSLGRNX whenever that structure is closed. The row contains the opening log relative byte address (RBA), the closing log RBA, or both for the structure. The log RBA is the relative byte address in the log data set where open and close information about the structure is contained.

The use of SYSLGRNX speeds up recovery by limiting the log information that must be scanned to apply changes to a table space or partition being recovered.

The two indexes defined on SYSLGRNX are DSNLLX01 (a clustered index) and DSNLLX02.

System Utilities: DB2 inserts a row in the system utilities table space (SYSUTILX) for every utility job that is run. The row remains there until the utility completes its full processing. If the utility terminates without completing, DB2 uses the information in the row to restart the utility. The indexes defined on SYSUTILX are DSNLUX01 and DSNLUX02.

Database Descriptor: The database descriptor table space (DBD01) contains internal control blocks, called *database descriptors* (DBDs), that describe the databases existing within DB2. 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. Figure 9 illustrates the contents of DBD01.

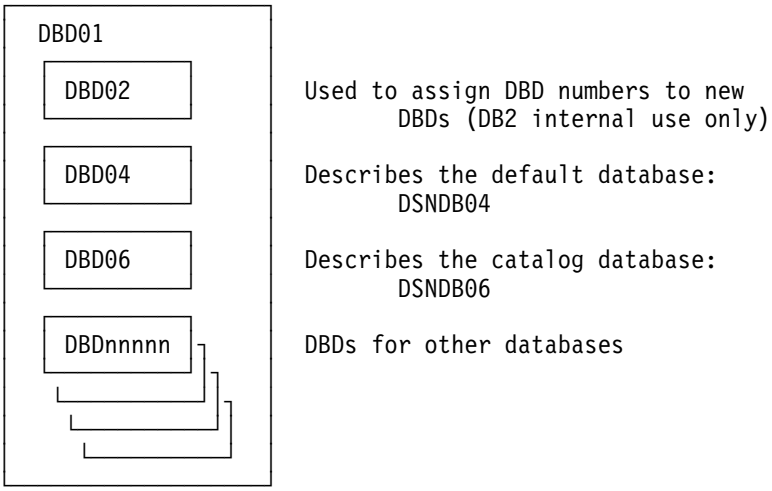


Figure 9. Contents of the Database Descriptor Table Space (DBD01). DBD02, DBD04, and DBD06 are shipped with DB2. Other DBDs are produced when databases are created.

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

The archive log can consist of up to 1000 data sets, each of which is a sequential data set (physical sequential) that resides on a DASD or magnetic tape volume. An archive log data set is created during the log off-load process (when an active log data set is copied to an archive log data set). The archive log can be cataloged in an integrated catalog facility catalog and protected with an MVS data set password or with resource access control facility (RACF).

DB2 allows you to 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.

For a detailed description of the logs, their contents, and the process of off-loading from active to archive logs, see “Chapter 4-3. Managing the Log and the Bootstrap Data Set” on page 4-83 .

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.

  For any log, the list tells the RBA range (or LRSN range in a data sharing environment) in each data set. (The list specifies separate log data set name entries for each volume on which the log resides.) For the active log, the list also tells which are full and which are available for reuse. DB2 records data about the log data set in the BSDS each time a new archive log data set is defined or an active log data set is reused.
- A wrap-around inventory of all recent DB2 checkpoint activity. DB2 uses this information during restart processing.
- The distributed data facility (DDF) communication record. This record contains the DB2 location name, the virtual telecommunications access method (VTAM) LU name, and the password used to connect DB2 to VTAM. DB2 uses this information to establish the distributed database environment.
- The bootstrap dataset (BSDS) also includes a table of IP addresses. These uniquely identify a host within the TCP/IP network.

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.

For a more complete description of the functions and uses of the BSDS, see “Managing the Bootstrap Data Set (BSDS)” on page 4-92 .

Buffer Pools

Buffer pools, also known as *virtual buffer pools*, are areas of virtual storage used temporarily to store pages of table spaces or indexes. When an application program needs to access a row of a table, DB2 retrieves the page containing that row and places the page in a buffer. If the row is changed, the buffer must be written back to the table space. If the needed data is already in a buffer, the application program will not have to wait for it to be retrieved from DASD. The result is faster performance. The sizes of virtual buffer pools can be changed while DB2 is running. The result is greater flexibility. See Chapter 2 of *Command Reference* for details about the ALTER BUFFERPOOL command.

DB2 supports a second level of storage for each virtual buffer pool if your system meets the following requirements:

- A processor that supports MVPG hardware instruction and Asynchronous Data Mover Facility licensed internal code
- Expanded storage available in ES/9000 hardware
- MVS/ESA Version 4 Release 3 with the Special Programming Enhancement containing support for Asynchronous Data Mover Facility

The second level of storage, the *hiperpool*, is an extension to the virtual buffer pool. Virtual buffer pools hold the most frequently accessed data. 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*, 2GB expanded storage areas. Using hiperspaces and hiperpools improves performance because you can cache up to 8GB to help avoid I/O operations.

When you install DB2, you make the following decisions about virtual buffer pools and hiperpools:

- Specify the number of 4KB virtual buffer pools and 32KB virtual buffer pools that you plan to use. You can have up to 50 4KB virtual buffer pools, and up to 10 32KB virtual buffer pools. The number of buffers within each pool is always less than or equal to the corresponding value specified on one of the buffer pool sizes panels (DSNTIP1, DSNTIP2).
- Specify whether you want a hiperpool to exist for a given virtual buffer pool. There can be only one hiperpool for each virtual buffer pool, and the sum of all hiperpools must not exceed 8GB of storage.

Another level of buffer pools is the group buffer pool, associated with the DB2 data sharing environment. For information relating to group buffer pools, see *Data Sharing: Planning and Administration*.

Communications Database

The *communications database* (CDB) is part of the catalog shipped with DB2. The distributed data facility (DDF) uses the CDB to map DB2 location names to VTAM LU names and TCP/IP IP addresses or domain names. The CDB also handles security translation requirements and communication requirements. The CDB is used to get information about communicating with other DB2 subsystems, or with remote locations that support Distributed Relational Database Architecture. Once your subsystem is installed, you must populate the CDB tables with unique information that enables your location to send and receive distributed data requests. For more information on the CDB, see *Installation Guide* .

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 statements to selected application identifiers (plans or collections of packages). Once this database is created, you must populate these tables to make use of this facility. The system name for this database is DSNRGFDB. For more information about DDL registration, see “Chapter 3-3. Controlling Access Through a Closed Application” on page 3-49 .

Resource Limit Facility Database

The *resource limit facility database* contains tables that limit the amount of processor time permitted for the execution of dynamic SELECT, UPDATE, DELETE, and INSERT SQL statements. You can establish a single limit for all users, different limits for individual users, or both. No limits apply to those with installation SYSADM or installation SYSOPR authority.

The limits are defined in one or more resource limit specification tables (RLST). One RLST is used for each invocation of the resource limit facility and is identified on the START RLIMIT command. If you are using DDF, the RLST contains columns so you can specify limits for primary authorization IDs, plan names, or package names from other subsystems. The RLST also governs BIND authority. You can have more than one RLST, but only one RLST is active at any one time. The system name for this database is DSNRLST.

| The RLST also contains a column that determines which mode of parallelism is
| disabled, if query CP parallelism or query I/O parallelism is possible. For more
| information on the RLST, see “Resource Limit Facility (Governor)” on page 5-76 .

Work File Database

The *work file database* is used as storage for processing SQL statements that require working space. You can create a work file database using the CREATE
| DATABASE and 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. One member of the data sharing group can have the name DSNDB07, but you can create a work file database with a more meaningful name.
| The default is DSN1. The recommended name is the DB2 subsystem name.

Control and Maintenance of DB2

DB2 is controlled and maintained by the following:

- Commands, which can be entered at a terminal or a MVS console
- Utility jobs, which run as standard MVS batch jobs.

Commands

DB2 is controlled by commands entered at a terminal or MVS console. 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.

For example, the command -START DB2 starts DB2. -STOP DB2 stops it. For a description of the steps used in operating DB2, and the commands used to implement them, see “ Section 4. Operation and Recovery” on page 4-1 of *Administration Guide* . For more information on commands, see Chapter 2 of *Command Reference*.

Utility Jobs

Many of the tasks of maintaining DB2 data, such as loading a table, copying a table space, or recovering a database to some previous point in time, are done by parts of DB2 called *utilities*.

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. To use DB2I, follow your local procedures for logging on to TSO, entering ISPF, and displaying the DB2I menu, shown in Figure 12 on page 1-40.

You control each operation by entering values that describe it on the panels provided. There are also help panels giving the syntax and examples of commands and utility control statements. To access the help panels, press the HELP PF key. (The HELP PF key can be set locally, but typically it is PF key 1.)

A utility control statement tells a particular utility what task to perform; many examples appear in this book. To run a utility job, first enter the control statement in a data set you use for input. Then invoke DB2I and select option 8, UTILITIES, on the DB2I Primary Option Menu. In some cases, you need other data sets; for example, the LOAD utility requires a data set containing the data that is to be loaded. For detailed instructions, see Section 1 of *Utility Guide and Reference* .

High Availability

It is not necessary to start or stop DB2 often. For nearly continuous operation, DB2 has been designed with the following capabilities:

- Online definition and modification of database and authorization descriptors.
- Online binding of application plans.
- Online changing of buffer pool and hiperpool sizes.
- Online execution of most utilities. For example:
  - You can recover online such structures as table spaces, partitions, data sets, a range of pages, a single page, and indexes.
  - You can recover several indexes or index partitions simultaneously to reduce recovery time.
  - You can read and update a table space while copying it.
  - You can reorganize table spaces and partitions separately. and read the data during the unload phase. You can specify the degree of access to your data during reorganization
- Availability of a table space (provided it is not explicitly stopped) after an I/O error, except for any portions that span the error ranges.
- Fewer pages are unavailable because of error conditions. For any structure that has a problem while applying a log record to the structure, DB2 adds the relevant pages and log ranges to the logical page list (LPL). This leaves only pages affected by the error condition unavailable.
- Improved reorganization time for a table with a low cluster ratio by specifying the SORTDATA parameter on the REORG utility.
- Using package versions permits binding while the applications continue to run. If there is a problem in the new application, the old program uses the old

version of the package. When an application changes, only the programs that have changed need to be rebound.

- Continuing operation of DB2 after an I/O error writing a log record. On the active log, it moves to the next data set; on the archive log, it dynamically allocates another data set.
- Remote site disaster recovery methods that allow you to prepare for disasters that could cause a complete shutdown of your local DB2 system.
- Typical continuation of DB2 during restoration of dual operation of the bootstrap data set, active logs, and archive logs if degradation to single copy mode was necessary.
- DB2's data sharing function allows applications running on more than one DB2 subsystem to read and write to the same set of data concurrently. For details on the benefits of data sharing see *Data Sharing: Planning and Administration*.

To reduce the probability and duration of unplanned outages you should periodically back up and reorganize your data. Because these affect the availability of the databases, you should limit your use of, and understand the options of, utilities such as COPY and REORG.

The CONCURRENT and SHRLEVEL options of the COPY utility can minimize outages and improve availability of DB2 data during backup processing. If you use the SHRLEVEL CHANGE option, the data sets being copied are available during the entire copy operation. If you use SHRLEVEL REFERENCE, the availability of data sets depends on whether you also use the CONCURRENT opiton, which invokes DFSMS Concurrent Copy.

- If you specify SHRLEVEL REFERENCE but not CONCURRENT, data sets are unavailable until the copy operation is complete.
- If you specify SHRLEVEL REFERENCE and CONCURRENT, data is unavailable only until DFSMSdss finishes logical processing of the list of data sets to be copied.

For information on using DB2 utilities such as COPY and REORG, see *Utility Guide and Reference*.

Unplanned outages are difficult to avoid entirely. However, the time that elapses because of an unplanned outage can be minimized, and the occurrence of these outages can be reduced. When unplanned outages occur, you can use the RECOVER utility to restore a damaged DB2 structure. For more information on recovering data, see “Recovering Table Spaces and Data Sets” on page 4-141 and Section 2 of *Utility Guide and Reference* .

It is important to avoid I/O errors on table spaces, indexes, logs, and the bootstrap data set because recovery of these errors (using the RECOVER utility) causes unplanned outages. For more information on I/O error recovery, see Section 2 of *Utility Guide and Reference*.

To ensure continuous availability, it is important to monitor the databases regularly. Monitoring measures the efficiency of your database, in both performance and space utilization. Most of your base tables and indexes are constantly being changed through updates, inserts, and deletions. Monitoring your space utilization can prevent problems. You can monitor and tune a database by using the RUNSTATS and STOSPACE utilities.

DB2 and the MVS Environment

DB2 operates as a formal subsystem of MVS/ESA. 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 requires several different address spaces for the following purposes:

- One for *database services*, DSN1DBM1, which manipulate most of the structures in user-created databases.
- One for *system services*, DSN1MSTR, which perform a variety of system-related functions.
- One for *distributed data facility*, DSN1DIST, which provides support for remote requests.
- One for the *internal resource lock manager* (IRLM), IRLMPROC, which controls DB2 locking.
- One for *DB2-established stored procedures*, DSN1SPAS, which provides an isolated execution environment for user-written SQL programs at a DB2 server.
- Zero to many for *WLM-established stored procedures* to be handled in order of priority and isolated from other stored procedures running in other address spaces.
- At least one, possibly several, of the following types of *user address spaces*:

  TSO
  Batch
  CICS
  IMS dependent region
  IMS control region

Figure 10 on page 1-37 shows how the address spaces relate to batch, TSO, IMS, and CICS user address spaces. Each user address space communicates with database services, system services, and distributed data facility address spaces. In addition, the IMS user address space communicates with the IRLM and IMS address spaces. The system services, database services, CICS, IRLM, IMS, stored procedures and distributed data facility address spaces communicate with each other as the arrows indicate.

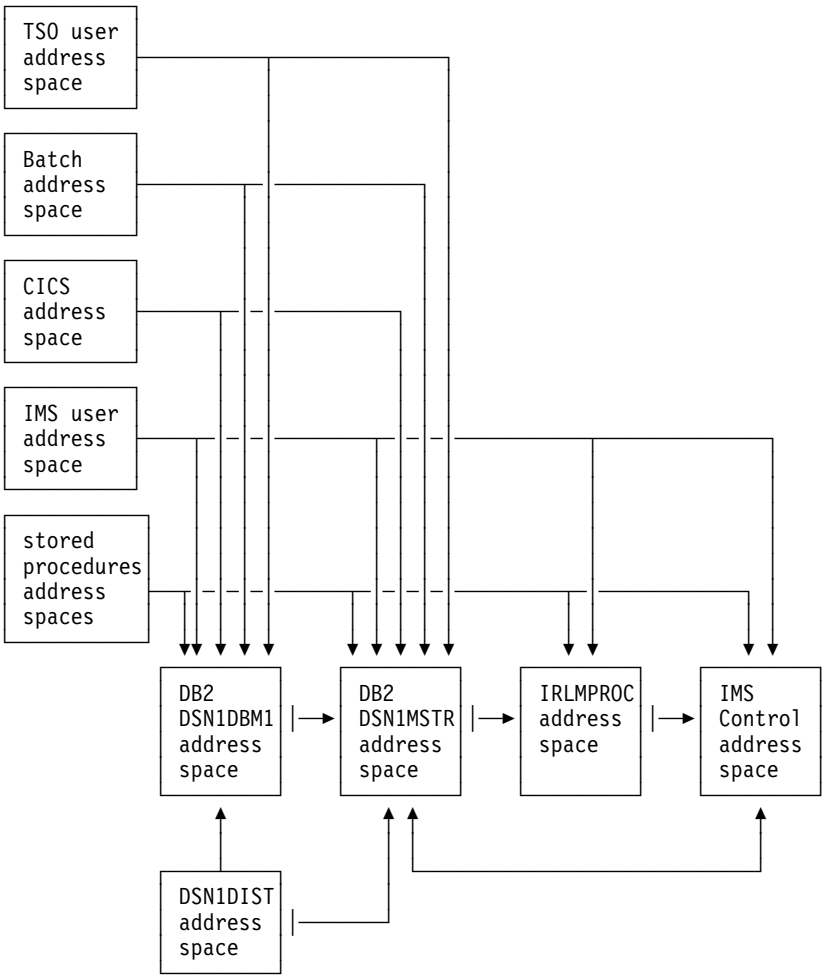


Figure 10. Relationship between DB2 Users and DB2-Related Address Spaces

DB2 and MVS

As a formal subsystem of MVS, DB2 uses:

- The MVS subsystem interface (SSI) protocols
- Key 7 operation and storage
- Synchronous cross-memory services for address space switching
- System Management Facilities (SMF) for statistics, accounting information, and performance data
- VTAM and TCP/IP for distributed data facility
- These reliability and serviceability features:

  Functional recovery routines (FRR)
  ESTAE recovery routines
  SYS1.LOGREC
  SYS1.DUMP

You can enter all DB2 commands from an authorized MVS console by using a subsystem command prefix (composed of 1 to 8 characters) at the beginning of the

command. The default subsystem command prefix is *-DSN1*. You can change the value when you install or migrate DB2.

DB2 and RACF

Resource Access Control Facility (RACF) can be used to control access to your MVS system. If you are using RACF Version 1 Release 9 or later, you can control access to the distributed data facility (DDF) based on the connecting partner's LU name. When users begin sessions with TSO, IMS, or CICS, their identities are checked to prevent unauthorized access to the system.

Recommendation: use RACF to run security checks on DB2 users and to protect DB2 resources. RACF 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 RACF. By using
| the access control authorization exit, a user can bypass some or most of DB2
| authorization checking. For more information on writing exit routines, see Appendix
| B of *Administration Guide*.

For a detailed description of security methods available in DB2, see " Section 3. Security and Auditing" on page 3-1.

DB2 and SMS

DFSMSdfp *storage management subsystem* (SMS) can be used to manage DB2 DASD data sets. Private data, image copies, and archive logs are possible candidates for space management with SMS. The decision to use SMS to manage DB2 data sets must be made with your site's storage administrator. For more information about using SMS to manage DB2 data sets, see *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide* .

Partitioned data set extended (PDSE), a feature of DFSMSdfp, provides a way to manage partitioned data sets. PDSE allows multiple access at a data set's member level rather than at the data set level, removing the concurrent access constraints of regular partitioned data sets. DB2 users who have MVS/ESA and DFSMSdfp installed should consider using PDSE data sets for their DBRM libraries.

| PDSE data sets must be managed by SMS and stored on direct access storage
| devices. See *MVS/DFP: Managing Non-VSAM Data Sets* for information about
| differences of PDS and PDSE data sets and how to convert them.

DB2 and TSO Attachment Facility

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.

Whether you access DB2 in foreground or batch, attaching through the TSO
attachment facility and the DSN command processor makes access easier. The
DSN command processor (DSN) executes as a TSO command processor. DB2

subcommands that execute under DSN are therefore subject to the command size
limitations as defined by TSO. See Appendix B in *TSO/E Programming Services* for
additional information on the limits for TSO/E services routines. TSO allows authorized DB2 users or jobs to create, modify, and maintain databases and application programs. DB2I invokes the DSN command processor before invoking the supported DSN subcommands shown in Figure 12 on page 1-40. 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, 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.

Figure 11 shows the relationship between DB2 and TSO as used by the DB2 interactive (DB2I) service.

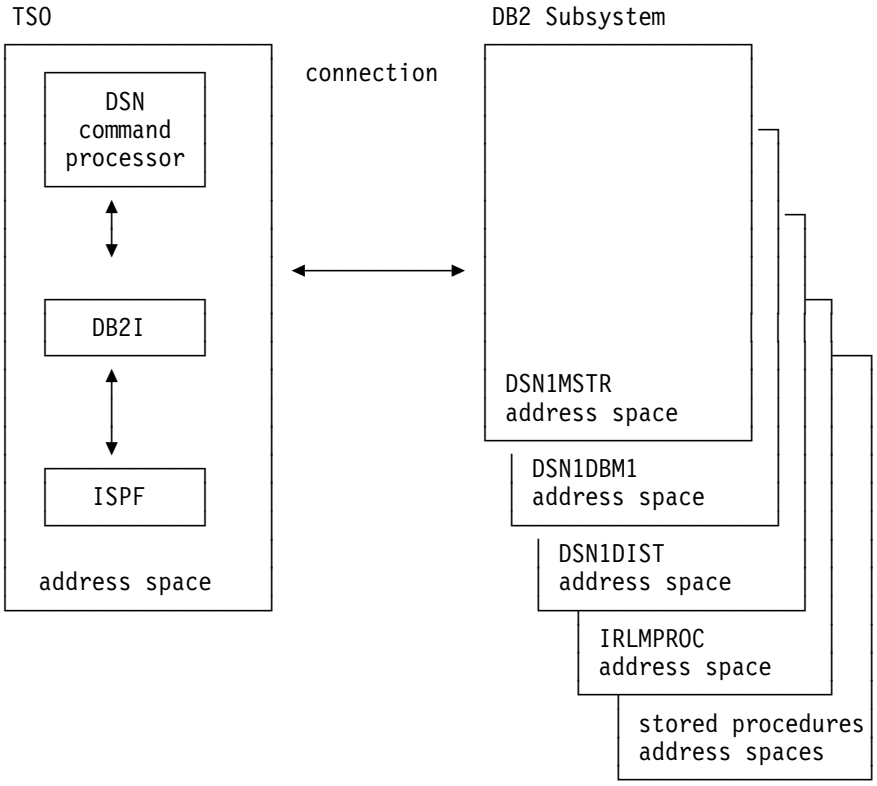


Figure 11. Relationship of DB2 to TSO

DB2 and ISPF

DB2 provides *Interactive System Productivity Facility* (ISPF) panels that allow you to perform most DB2 tasks interactively. These panels make up a DB2 facility called DB2 interactive (DB2I). Figure 12 on page 1-40 provides an example of a DB2I ISPF panel.

```
                          DB2I PRIMARY OPTION MENU
===>_

Select one of the following DB2 functions and press ENTER.

 1 SPUFI                  (Process SQL statements)
 2 DCLGEN                 (Generate SQL and source language declarations)
 3 PROGRAM PREPARATION    (Prepare a DB2 application program to run)
 4 PRECOMPILE             (Invoke DB2 precompiler)
 5 BIND/REBIND/FREE       (BIND, REBIND, or FREE plans or packages)
 6 RUN                    (RUN an SQL program)
 7 DB2 COMMANDS           (Issue DB2 commands)
 8 UTILITIES              (Invoke DB2 utilities)
 D DB2I DEFAULTS          (Set global parameters)
 X EXIT                   (Leave DB2I)


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

Figure 12. DB2I Primary Option Menu

Because application programs are invoked under the TSO terminal monitor program, you can use ISPF to structure input and output. For information on using DB2 and ISPF together in an application, see Section 6 of *Application Programming and SQL Guide* .

Call Attachment Facility

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 supplied by CAF. For more information on CAF, see Section 6 of *Application Programming and SQL Guide*.

DB2 and CICS

The *Customer Information Control System* (CICS) attachment facility provided with DB2 allows you to 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.

The CICS attachment facility uses standard CICS command-level services where needed; for example, 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. These calls affect the application in the standard CICS way.

You can use DB2 with an extended recovery facility to facilitate recovery from a CICS failure. To accomplish this, you must place all DB2 data sets on DASD shared between the primary and alternate XRF systems. This enables DB2 to be manually stopped on the primary system and started on the alternate system. You must then ensure that the DB2 data sets on the shared DASD cannot be updated at the same time by both the primary and alternate XRF systems. You might use a multi-system DASD serialization function, such as Global Resource Serialization (GRS), or any other means of shared DASD protection. For more information about XRF, see “Extended Recovery Facility (XRF) Toleration” on page 4-160 and *IMS/ESA Administration Guide: System*. For more information on global resource serialization, see *MVS/ESA Planning: Global Resource Serialization*.

Figure 13 shows the relationship between DB2 and CICS. For a detailed discussion of the connections, see “Chapter 4-2. Monitoring and Controlling DB2 and Its Connections” on page 4-23 of *Administration Guide* .

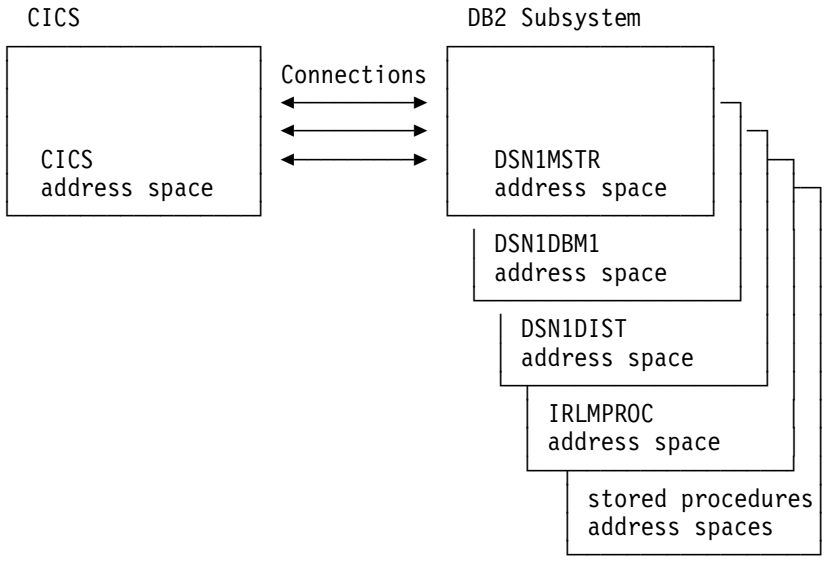


Figure 13. Relationship of DB2 to CICS

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.

There are significant changes to the CICS attachment facility with CICS Version 4. For more information on using CICS with DB2 see *Installation Guide* .

DB2 and 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.

You can use DB2 with an extended recovery facility to facilitate recovery from an IMS failure. To accomplish this, you must place all DB2 data sets on DASD shared between the primary and alternate XRF systems. This enables DB2 to be manually stopped on the primary system and started on the alternate system. You must then ensure that the DB2 data sets on the shared DASD cannot be updated at the same time by both the primary and alternate XRF systems. You might use a multi-system DASD serialization function, such as Global Resource Serialization (GRS), or any other means of shared DASD protection. For more information about XRF, see “Extended Recovery Facility (XRF) Toleration” on page 4-160 and *IMS/ESA Administration Guide: System*. For more information on global resource serialization, see *MVS/ESA Planning: Global Resource Serialization*.

Figure 14 on page 1-43 shows the relationship between DB2 and IMS.

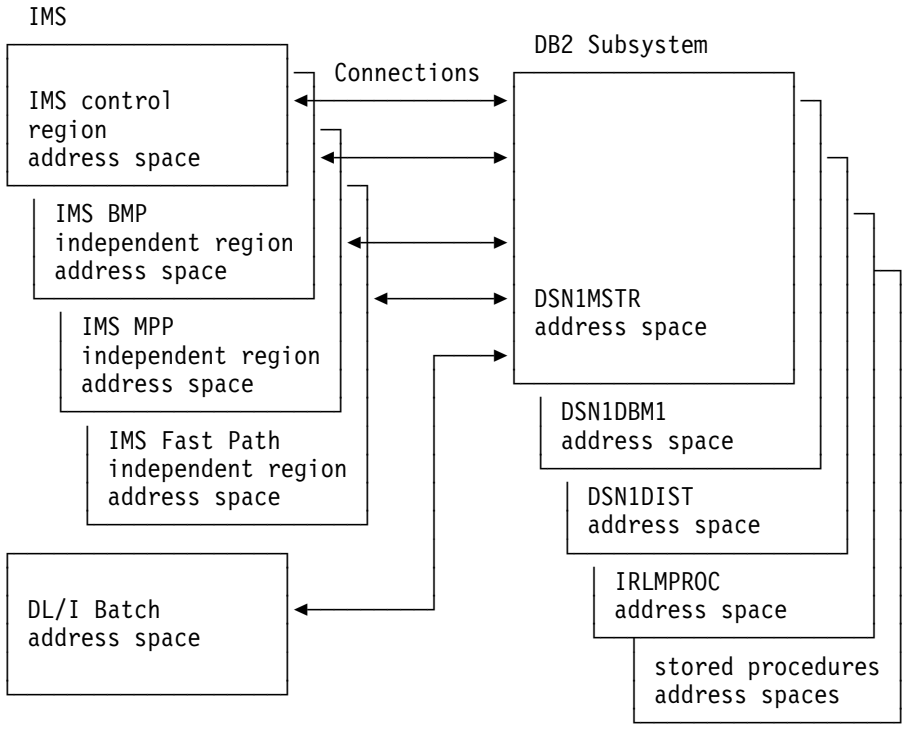


Figure 14. Relationship of DB2 to IMS

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.

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 Data Propagator NonRelational (DPropNR) licensed program. Data Propagator NonRelational Release 1 allows automatic updates to DB2 tables when corresponding information in an IMS DB (database) is updated. Data Propagator NonRelational Release 2 adds the ability to automatically update an IMS DB when corresponding information in your DB2 tables is updated.

Data Propagator NonRelational Release 2 also lets application programs access up-to-date IMS or DB2 changes without requiring users to convert production

applications or periodically copy data back and forth between DB2 tables and IMS DBs.

See *DataPropagator NonRelational MVS/ESA Administration Guide* for more information about Data Propagator NonRelational.

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.

DB2 and DL/I Batch

The DL/I batch support allows you to access both IMS data (DL/I) and DB2 data in the IMS batch environment. DL/I batch access allows:

- 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.

For more information on DL/I batch, see Section 5 of *Application Programming and SQL Guide*.

DB2 and DDF

The *distributed data facility* (DDF) is an optional feature that allows a DB2 application to access data at other DB2s and at remote relational database systems that support IBM's Distributed Relational Database Architecture (DRDA). In addition, DDF allows applications running in a remote application requester environment that supports DRDA to access data in DB2 subsystems. Figure 15 on page 1-45 gives an overview of DDF support.

Substantial improvements in distributed database access have been achieved with the stored procedures function and distributed threads enhancements.

Stored procedures solve the problem of high processor and elapsed time costs that DRDA users experience during SQL processing when accessing data managed by DB2 for OS/390. This function introduces an SQL interface that allows an SQL requester to invoke user-written SQL programs, or *stored procedures*, at a DB2 server. Local DB2 applications or remote DRDA applications can issue the new SQL CALL statement to invoke a stored procedure. With a single send or receive operation, a series of SQL statements are invoked in the stored procedure, thus significantly decreasing the costs of distributed SQL statement processing.

Distributed threads enhancements allow you to have up to 25000 distributed threads connected to DB2 at the same time. This increase gives you room to grow your distributed applications; more applications can now connect to DB2 without delays.

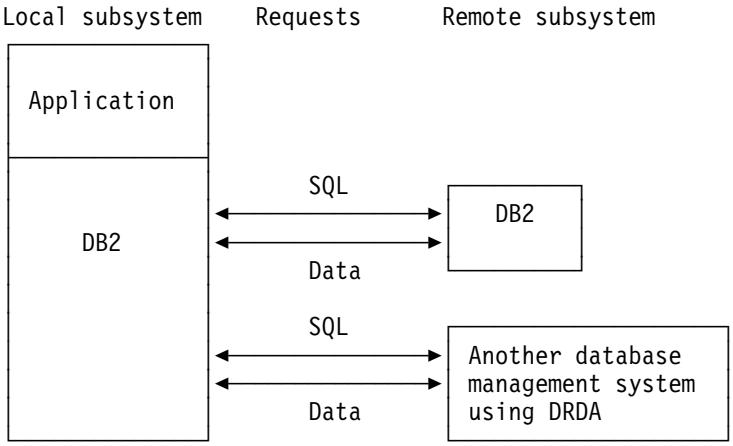


Figure 15. Overview of Distributed Database Support

The decision to access distributed data has implications for many DB2 activities: application programming, data recovery, authorization, and so on. For a discussion of these implications, see “Plan for Distributed Data” on page 2-9 .

Data Sharing

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.

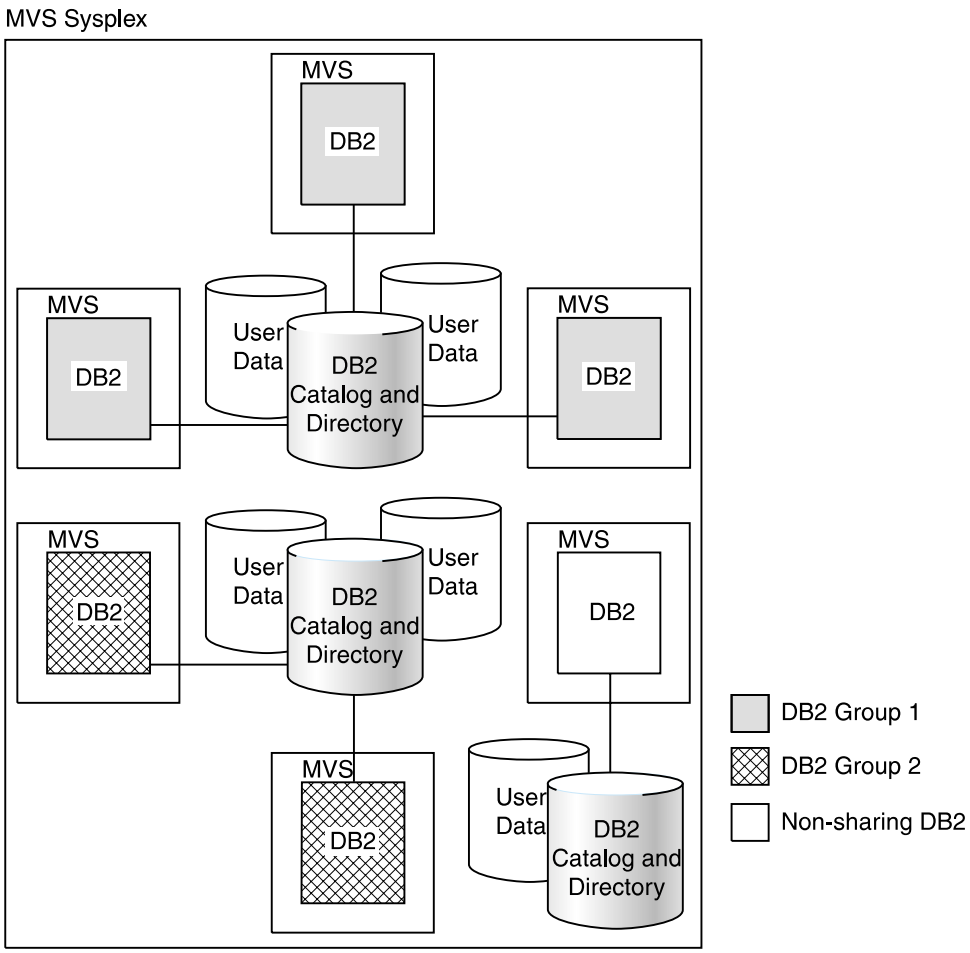


Figure 16. Data Sharing

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. DB2 data sharing is intended as a replacement for shared read-only data. Unlike read-only data sharers, all members of a data sharing group have equal and concurrent read and write access to databases.

Data sharing allows you to add another DB2 onto another central processor complex and access the same data through the new DB2. There is no need to manage copies or distribute data. All DB2s in the group have concurrent read and write access, and a single catalog and a single directory are used by all.

Data sharing allows you great flexibility in environment configuration. You can use separate MVS images, each tailored for its user set, sharing the same data. You can build your system incrementally as need dictates and set up members for use during peak times only.

For more information, see *Data Sharing: Planning and Administration*.

Section 2. Designing a Database

Chapter 2-1. Designing a Database

This chapter provides an overview of the other chapters in this section and introduces some DB2-specific terminology. Additionally, this chapter briefly describes the two types of database design, *logical* and *physical*.

Using the Design Chapters

This section of the *Administration Guide*:

- Introduces database design terminology
- Describes relational database design concepts
- Provides planning information for logical design activities
- Details DB2 database design implementation activities
- Tells how to use the catalog as a design tool
- Describes how to alter your implemented design

"Chapter 2-2. Designing Tables and Views" on page 2-11 explores the elements of table design in relational databases, including the types of relationships, normalization and denormalization and also considers views and reasons for using views of tables.

"Chapter 2-3. Maintaining Data Integrity" on page 2-19 gives an overview of how to ensure that rows are unique, maintain referential integrity, and ensure data validity.

"Chapter 2-4. Designing Columns" on page 2-39 describes how to design columns in tables, define primary and foreign keys, and explains the types of data you can put in columns.

"Chapter 2-5. Designing Indexes" on page 2-51 explains the differences between the two types of indexes used in DB2, and how and when you should use them. Designing index spaces is also described in this chapter.

"Chapter 2-6. Designing Table Spaces" on page 2-59 provides information about simple, segmented, and partitioned table spaces, and describes how to compress data in DB2 databases.

"Chapter 2-7. Designing Storage Groups and Managing DB2 Data Sets" on page 2-67 explains how to manage DB2 data sets through use of storage groups.

"Chapter 2-8. Designing a Database in a Distributed Environment" on page 2-73 provides information necessary for designing databases in a distributed environment, and describes design considerations for stored procedures.

"Chapter 2-9. Implementing Your Design" on page 2-79 details specific implementation activities, such as creating storage groups, table spaces, databases, and tables.

"Chapter 2-10. Loading Data into DB2 Tables" on page 2-113 is an overview of how data is loaded into DB2 tables.

"Chapter 2-11. Using the Catalog in Database Design" on page 2-117 describes how to use the catalog to find information about DB2 databases.

"Chapter 2-12. Altering Your Database Design" on page 2-123 tells how you can alter your databases after you have created them.

Terminology

This section defines our basic terms for database design; other terms are defined later as they occur. The terms used in this book might be used differently in other non-DB2 books.

The term *database* is not used consistently in the data processing industry. Sometimes, database means the entire set of data used by an enterprise, whether the data is all computer-processed or not. Elsewhere, database means all the machine-readable data managed by a particular computer application program. And particular products, like DB2, use database in specialized senses. In this book, database is used in the following ways:

- A *relational database* is the entire set of data managed by one instance of a relational database management system (DBMS), like DB2. Where "relational database" is clear from context, it is shortened to *database*. In this chapter, "database" generally means "relational database".
- A *DB2 database* is a DB2 object created by the CREATE DATABASE statement. Where "DB2 database" is clear from the context, it is shortened to *database*. For example, "database DSNDB06" refers to a named DB2 object.

An *entity* is a person, object or concept about which you wish to store information. In DB2, information about entities are stored in *tables*, which are described in "Chapter 2-2. Designing Tables and Views" on page 2-11.

Some of the entities described in the sample DB2 tables are employees, departments, and projects. (See "Appendix A. DB2 Sample Tables" on page X-7, for a description of the sample database.) In the sample employee table (DSN8510.EMP), the employee "entity" has attributes, or *properties*, such as employee number, job code, birth date, and salary amount. Those properties appear as the columns EMPNO, JOBCODE, BIRTHDATE, and SALARY. An *occurrence* of the entity "employee" consists of the values in all of the columns for one employee. Each employee has a unique employee number (EMPNO) that can be used to identify an occurrence of the entity "employee."

In DB2 tables, entities and properties of similar entities are represented as columns, and occurrences are represented as values in the columns, as in the following table:

Table 2. Occurrences and Properties of an Entity

Entity	◄ Properties ►			
(employee)	**EMPNO**	**JOBCODE**	**BIRTHDATE**	**SALARY**
Sally Kwan	000030	60	1941-11-05	38250
John Geyer	000050	58	1925-09-15	40175

Logical Design

A database is more than a collection of employee and department numbers, parts and inventory identifiers, and dollars and cents. A database is a representation of the people and things your business needs to operate, and the way those people and things relate to each other.

The logical structure of data is the entities and their relations to each other, while the physical data structure is the software implementation of the entities and their relations. Relationships between dissimilar entities are represented by like values in columns of different tables.

Logical design is the process of listing entities and mapping their relationships.

A Sample List of Entities

For example, your business has employees, in departments with managers, who produce goods and services, using parts and inventory, to fill orders placed by customers. One listing of the entities is:

- Employees
- Departments
- Managers
- Parts
- Orders
- Customers

Listing the types of entities is the first, and often easiest step in logical database design. Tracing the relationships of entities to each other can be complex, because there is often more than simple one-to-one causal correspondence between the entities and their associated properties.

For example, employees are usually assigned unique employee numbers and report to one department. But managers can manage more than one department, and employees can be working on several different products, filling multiple orders from a variety of customers. Customers, also usually assigned a unique customer number, can be ordering several products at one time.

DB2 Structures

In DB2, you use different structures (also called objects) to turn your logical database design into the physical database. You create these structures, and they determine how the data is accessed and stored. This section briefly describes the DB2 structures that require storage allocation. More detailed information about creating DB2 structures can be found in “Chapter 2-9. Implementing Your Design” on page 2-79 and *SQL Reference*.

Tables: Tables are the primary means of organizing entities in DB2. Tables are a collection of unordered rows, each representing a specific entity. Table 2 on page 2-6 is an example of a table containing employee information. In addition, you can use temporary tables when your database design only needs a table for the life of an application process.

Columns: Columns contain the properties of the entities in your tables. In Table 2 on page 2-6, the properties of the employees are their employee numbers, job

codes, birthdays, and salaries. You name the columns of your table as you create or alter your table.

Indexes: Indexes are a set of pointers you specify that provide access to data stored in tables. Indexes are defined separately from tables. Indexes are used to:

- Ensure uniqueness
- Provide fast access to data

Table Spaces: A table space is one or more data sets used to store one or more tables. In DB2, there are several types of table spaces. More information about the types of table spaces can be found in “Chapter 2-6. Designing Table Spaces” on page 2-59.

Index Spaces: Index spaces are the implicitly-created physical storage areas for indexes. Each index gets an individual index space.

Storage Groups: Storage groups is a list of DASD volumes you specify to hold your DB2 objects. Storage is then allocated from these volumes as your tables are loaded with data.

Databases: A database is a collection of table spaces and index spaces, and the data contained within them.

Physical Design

Using objects to create physical data structures is one part of physical design. Other considerations include appropriate organization of your data, allocating storage across one or more DASD volumes at one or more locations, the number of users who will need access to the data and what kind of access they will need, and the types of applications used to access the data.

Plan for Maintaining Data Integrity

The condition of a set of tables in which all references from one table to another are valid is called *referential integrity*; referential integrity is the enforcement of all referential constraints. Of course, having referential integrity does not mean that the data is necessarily correct. That the employee table shows every employee assigned to a valid department number is one thing; to have it show every employee assigned to the correct department is quite another.

When you design your tables, make sure that entities that refer to entities in other tables refer to entities and tables that actually exist. Referential integrity is described in “Chapter 2-3. Maintaining Data Integrity” on page 2-19.

Parent Key

The *parent key* of a table is either a primary key or a unique key that is part of a referential constraint.

Primary Key

The *primary key* of a table is the column or set of columns that provide the unique identifiers of the rows. For example, the primary key of the department table is the department number; the primary key of the project activity table is a combination of the project number, the activity number, and the activity starting date.

Foreign Key

A column or set of columns that refer to a parent key (it is common for the parent key to be a primary key) is called a *foreign key*. For example, the column of department numbers in the employee table is a foreign key because it refers to the primary key of the department table. The combination of project number, activity number, and activity starting date that appears in the employee to project activity table is a foreign key because it refers to the same combination of columns in the project activity table. The employee number in that same table is another foreign key; it refers to the primary key of the employee table.

Parent and Dependent Tables and Rows

The table containing the primary key is called the *parent table* and the one containing the foreign key is the *dependent table*. A dependent of a dependent is a *descendent*.

These terms should not suggest a “family tree” of dependencies. It is possible for a table to be a dependent of itself (called a *self-referencing* table), for two tables to be dependents of each other, or for there to be a cycle of tables each dependent on the one before it. A table can be the parent of many dependents, and it can also be a dependent of many parents. Refer to Figure 27 on page 2-19 for examples of various relationships.

Similar terms apply to rows. A row of a parent table that is referred to by some row of the dependent table is a *parent row*. The row that refers to it is a *dependent row*. But a row of a parent table is not always a parent row—perhaps nothing refers to it. Likewise, a row of a dependent table is not always a dependent row—the foreign key could allow null values, which refer to nothing.

Table Check Constraints

Table check constraints allow you to specify what values of a column in a table are valid, such as ensuring that phone numbers have a numeric value between 0000 and 9999. With table check constraints, you do not have to enforce constraints within an application program or with a validation routine.

Plan for Distributed Data

Distributed data is data processing in which some or all of the data is stored in more than one system. This section is an overview to help you evaluate whether distributed data is appropriate for your environment. You can find more information about distributed data in *Distributed Relational Database Architecture: Evaluation and Planning Guide*. “Chapter 2-8. Designing a Database in a Distributed Environment” on page 2-73 tells you how to set up DB2 for distributed data.

Plan for Data Security

The final step in logical database design is to plan how you will control access to the DB2 subsystem and its data. To find out what security mechanisms and choices are available to you, see “ Section 3. Security and Auditing” on page 3-1.

Chapter 2-2. Designing Tables and Views

This chapter explains the different types of relationships between data, the forms of tables, and normalization, and also considers creating views and reasons for using views of your tables.

Decide What Data to Record in the Relational Database

In a table, each column of a row is related in some way to all the other columns of that row. Some of the relationships expressed in the sample tables are:

- Employees are assigned to departments.

  Dolores Quintana is assigned to Department C01.

- Employees earn money.

  Dolores earns $23,800 per year.

- Departments report to other departments.

  Department C01 reports to Department A00.
  Department D01 reports to Department A00.

- Employees work on projects.

  Dolores and Heather work on projects IF1000 and IF2000.

- Employees manage departments.

Before you design your tables, you must understand entities and their relationships. “Employee” and “department” are entities; Sally Kwan is part of an occurrence of “employee,” and C01 is part of an occurrence of “department.” Entities and their relationships can be represented as in Table 3.

Table 3. Relationships in the DB2 Sample Tables

Entity	Relationship	Entity
Employees	are assigned to	departments
Employees	earn	money
Departments	report to	departments
Employees	work on	projects
Employees	manage	departments

The same relationship applies to the same columns in every row of a table. For example, one row of a table expresses the relationship that Sally Kwan manages Department C01; another, the relationship that John Geyer manages Department E01.

Define Tables for Each Type of Relationship

In a relational database, you can express several types of relationships. Consider the possible relationships between employees and departments. A given employee can work in only one department; this relationship is *single-valued* for employees. On the other hand, one department can have many employees; the relationship is *multivalued* for departments. The relationship between employees (single-valued) and departments (multivalued) is a *one-to-many* relationship. Relationships can be one-to-many, many-to-one, one-to-one, or many-to-many.

The type of a given relationship can vary, depending on the specific environment. If employees of some company belong to several departments, the relationship between employees and departments is many-to-many.

You will want to define separate tables for different types of relationships.

One-to-Many and Many-to-One Relationships

To define tables for each one-to-many and many-to-one relationship:

- Group all the relationships for which the “many” side of the relationship is the same entity.
- Define a single table for all the relationships in a group.

In Table 4, the “many” side of the first and second relationships is “employees” so we define an employee table (DSN8510.EMP).

Table 4. Many-to-One Relationships

Entity	Relationship	Entity
Employees	are assigned to	departments
Employees	earn	money
Departments	report to	(administrative) departments

The “many” side of the third relationship is “departments,” so we define a department table (DSN8510.DEPT). Figure 17 illustrates the process.

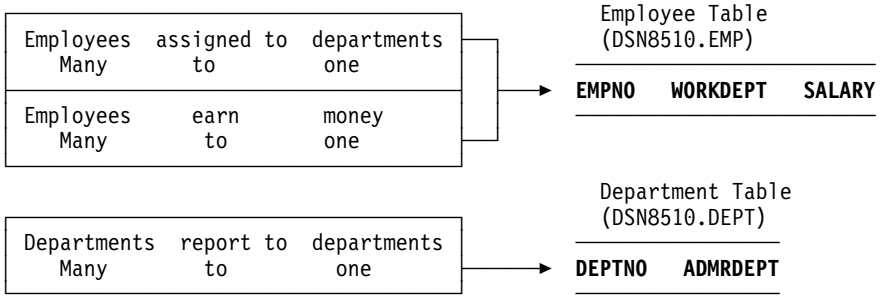


Figure 17. Assigning Many-to-One Facts to Tables

Many-to-Many Relationships

A relationship that is multivalued in both directions is a many-to-many relationship. An employee can work on more than one project, and a project can have more than one employee assigned. The questions “What does Dolores Quintana work on?” and “Who works on project IF1000?” both yield multiple answers. A many-to-many relationship can be expressed in a table with a column for each entity (“employees” and “projects”), as shown in Figure 18.

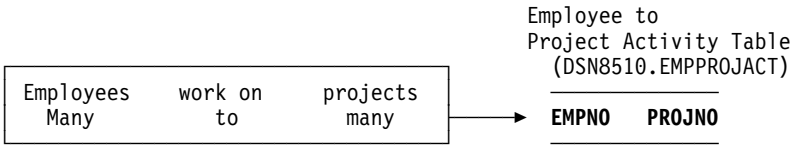


Figure 18. Assigning Many-to-Many Facts to a Table

One-to-One Relationships

One-to-one relationships are single-valued in both directions. A manager manages one department; a department has only one manager. The questions, “Who is the manager of Department C01?” and “What department does Sally Kwan manage?” both have single answers. The relationship can be assigned to either the department table or the employee table. Because all departments have managers, but not all employees are managers, it is most logical to add the manager to the department table as shown in Figure 19.

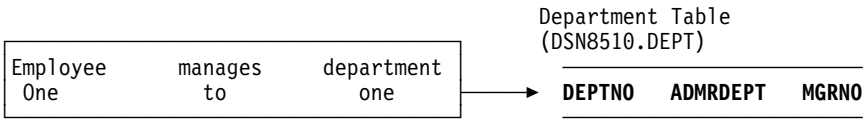


Figure 19. Assigning One-to-One Facts to a Table

Normalize Your Tables to Avoid Redundancy

Normalization helps you avoid redundancies and inconsistencies in your data. This section briefly reviews the rules for first, second, third, and fourth normal forms of tables, and describes some reasons why they should or should not be followed. The fifth normal form of a table, which is covered in many books on database design, is not described here.

Here are brief descriptions of the normal forms presented later:

Form	Description
First	At each row and column position in the table there exists one value, never a set of values.
Second	Each column that is not in the key provides a fact that depends on the entire key.
Third	Each nonkey column provides a fact that is independent of other nonkey columns and depends only on the key.
Fourth	No row contains two or more independent multivalued facts about an entity.

First Normal Form

Any relational table satisfies the requirement of first normal form: at each row-and-column position in the table there exists one value, never a set of values.

Second Normal Form

A table is in second normal form if each column that is not in the key provides a fact that depends on the entire key.

Second normal form is violated when a nonkey column is a fact about a subset of a composite key, as in the following example. An inventory table records quantities of specific parts stored at particular warehouses; its columns are shown in Figure 20.

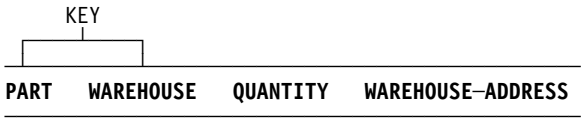


Figure 20. Key Violates Second Normal Form

Here, the key consists of the PART and the WAREHOUSE columns together. Because the column WAREHOUSE-ADDRESS depends only on the value of WAREHOUSE, the table violates the rule for second normal form. The problems with this design are:

- The warehouse address is repeated in every record for a part stored in that warehouse.
- If the address of the warehouse changes, every row referring to a part stored in that warehouse must be updated.
- Because of the redundancy, the data might become inconsistent, with different records showing different addresses for the same warehouse.
- If at some time there are no parts stored in the warehouse, there might be no row in which to record the warehouse address.

To satisfy second normal form, the information shown above would be in two tables, as in Figure 21.

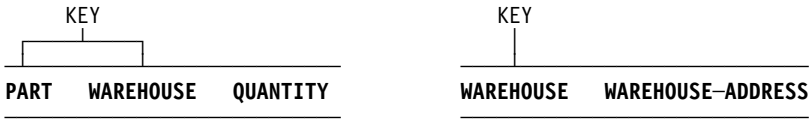


Figure 21. Two Tables Satisfy Second Normal Form

However, there is a performance consideration in having the two tables in second normal form. Application programs that produce reports on the location of parts must join both tables to retrieve the relevant information.

To better understand performance considerations, see “Consider Denormalizing Your Tables for Performance” on page 2-16.

Third Normal Form

A table is in third normal form if each nonkey column provides a fact that is independent of other nonkey columns and depends only on the key.

Third normal form is violated when a nonkey column is a fact about another nonkey column. For example, the first table in Figure 22 contains the columns EMPNO and WORKDEPT. Suppose a column DEPTNAME is added. The new column depends on WORKDEPT, whereas the primary key is the column EMPNO; thus the table now violates third normal form.

Changing DEPTNAME for a single employee, John Parker, does not change the department name for other employees in that department. The inconsistency that results is shown in the updated version of the table in Figure 22.

The table can be normalized by providing a new table, with columns for WORKDEPT and DEPTNAME. In that case, an update like changing a department name is much easier—the update only has to be made to the new table. An SQL query that shows the department name along with the employee name is more complex to write because it requires joining the two tables. This query will probably also take longer to execute than the query of a single table. In addition, the entire arrangement takes more storage space because the WORKDEPT column must appear in both tables.

Employee–Department Table (EMPDEPT) Before Update

EMPNO	FIRSTNME	LASTNAME	WORKDEPT	DEPTNAME
000290	JOHN	PARKER	E11	OPERATIONS
000320	RAMLAL	MEHTA	E21	SOFTWARE SERVICES
000310	MAUDE	SETRIGHT	E11	OPERATIONS

Employee–Department Table (EMPDEPT) After Update

EMPNO	FIRSTNME	LASTNAME	WORKDEPT	DEPTNAME
000290	JOHN	PARKER	E11	INSTALLATION MGMT
000320	RAMLAL	MEHTA	E21	SOFTWARE SERVICES
000310	MAUDE	SETRIGHT	E11	OPERATIONS

Figure 22. Update of an Unnormalized Table. Information in the table has become inconsistent.

Fourth Normal Form

A table is in fourth normal form if no row contains two or more independent multivalued facts about an entity.

Consider these entities: employees, skills, and languages. An employee can have several skills and know several languages. There are two relationships, one between employees and skills, and one between employees and languages. A table is not in fourth normal form if it represents both relationships, as in the example of Figure 23 on page 2-16.

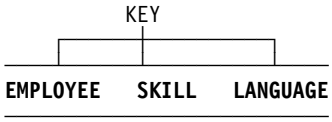


Figure 23. A Table That Violates Fourth Normal Form

Instead, the relationships should be represented in two tables, as in Figure 24.

Figure 24. Tables in Fourth Normal Form

If, however, the facts are interdependent—that is, the employee applies certain languages only to certain skills—then the table should *not* be split.

Any data can be put into fourth normal form. A good rule when designing a database is to arrange all data in tables in fourth normal form, and then decide whether the result gives you an acceptable level of performance. If it does not, you are at liberty to denormalize your design.

Consider Denormalizing Your Tables for Performance

"Normalize Your Tables to Avoid Redundancy" on page 2-13 describes normalization only from the viewpoint of logical database design. This is appropriate because the rules of normalization do not consider performance. See the example, given in "Second Normal Form" on page 2-14, of the column that contains the addresses of warehouses. The column is first shown as part of a table that contains information about parts and warehouses. To further normalize the design, the column is removed from that table and defined as part of a table that contains information only about warehouses. The other possible design (in which the column is part of both tables) is not considered, because the context of the discussion is logical database design.

What if there are applications that require information about both parts and warehouses, including the addresses of warehouses? The premise of the normalization rules is that the information can be retrieved by SQL statements that join the two tables. The problem is that a join operation can be time consuming, even for only two tables. As the number of tables increases, the access costs can increase enormously, depending on the size of the tables, the available indexes, and so on. For example, if indexes are not available, the join of many large tables could conceivably take hours. Furthermore, the number of tables that can be joined is at most 15 and, depending on the complexity of the statement, can be significantly less. Thus, a denormalized design might be absolutely necessary.

Consider the design in which both tables have a column that contains the addresses of warehouses. If this design makes join operations unnecessary, it could be a worthwhile redundancy. Addresses of warehouses do not change often. And if one does change, SQL makes it easy to update all occurrences.

Normalizing tables is recommended, but there can be performance problems, if, for example, user queries view data that is in more than one table, causing too many joins. It may become necessary for you to denormalize your tables.

What you have to consider is the trade-off—whether duplication, in several tables, of often requested columns is less expensive than the time it takes to perform joins. This duplication of columns in multiple tables is denormalization, and increases redundancy.

In the following tables, information about parts, warehouses, and warehouse addresses are contained in two tables, both in normal form.

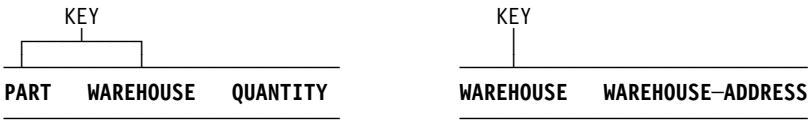


Figure 25. Two Tables Satisfy Second Normal Form

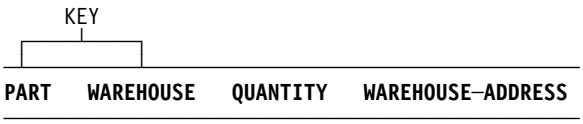


Figure 26. Denormalized Table

If you decide to denormalize, thoroughly document your denormalization. Describe, in detail, the logic behind the denormalization and the steps you took. Then, if in the future it becomes necessary to normalize again, an accurate record is available for those who must do the work.

Consider Creating Views of Your Tables

Some of your users might find that no single table contains all the data they need; rather, the data might be scattered among several tables. Furthermore, one table might contain more data than they want to see, or more than they should be authorized to see. For those situations, you can create *views*. A view is an alternative way of describing data that exists in one or more tables.

You can create a view any time after creating the underlying tables. The owner of a set of tables implicitly has the authority to create a view on them, and a user with SYSADM authority can create a view for any owner on any set of tables.

Reasons for Using Views

Some reasons you might want to use views are:

- To provide a customized table for a specific user

  Some tables might have a large number of columns, not all of which are of interest to all users. You can, in effect, create a smaller table for certain users by defining a view containing only the columns of interest.

- To limit access to certain kinds of data

  You can create a view containing only selected columns and rows from a table or tables. Users with the SELECT privilege on the view see only the information you describe. For example, a view could be defined that showed only the FIRSTNME, LASTNAME, WORKDEPT, and EDLEVEL columns for employees in Department D11.

- To allow you to alter tables without affecting application programs

For example, an application program that uses INSERT into T1 without a specified list of column names causes an error after you add a column to table T1. The error is generated because the number of values being inserted into the table is different from the number of columns in the table. If T1 is a view, you are protected from that error because adding a column to the table does not affect the view definition and, therefore, does not affect the application program.

When designing views, there are several restrictions you need to consider. See Chapter 6 of *SQL Reference*.

Using Joins

You can create a view that combines information from two or more tables by naming more than one table in the FROM clause. See “Creating a View Combining Information from Several Tables” on page 2-106 for more information.

Chapter 2-3. Maintaining Data Integrity

This chapter discusses the following topics:

Maintaining Referential Integrity

Throughout this section, you might want to refer to the diagram of tables and relationships in Figure 27.

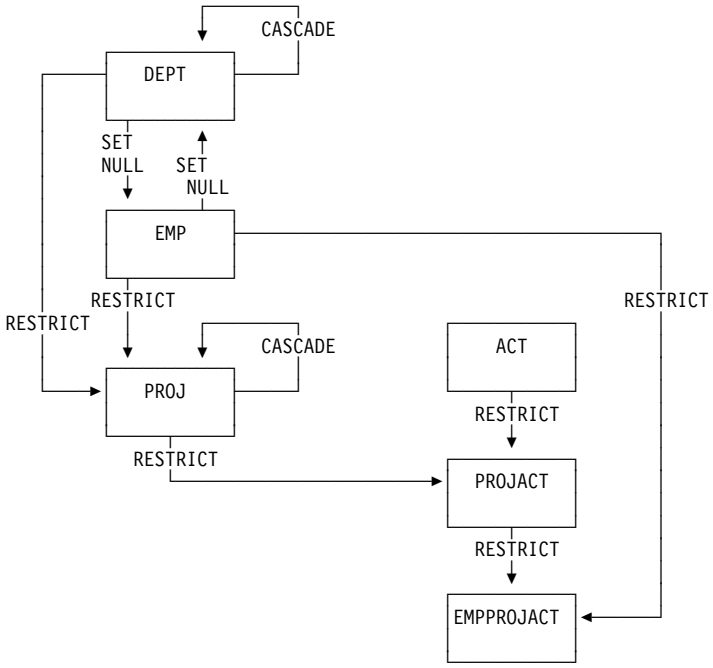


Figure 27. Relationships Among Tables in the Sample Application. Arrows point from parent tables to dependent tables.

A table can serve as the "master list" of all occurrences of an entity. In the sample application, the employee table serves that purpose for employees; only the numbers that appear in that table are valid employee numbers. Likewise, the department table provides a master list of all valid department numbers; the project activity table provides a master list of activities performed for projects; and so on.

There are times when normalizing a table means that one table must refer to other entities, thus creating occurrences of duplicate data. In the sample tables, for example, the employee table refers to departments by listing the department for which each employee works.

Clearly, when a table refers to an entity for which there is a master list, it should identify an occurrence of the entity that actually appears in the master list; otherwise, either the reference is invalid or the "master list" is incomplete.

You can let DB2 enforce referential integrity, or you can have your application programs enforce it. In either case, you must consider how a change to a table can be affected by its parent tables or might affect its dependent tables.

Identify One or More Columns as a Parent Key

If every row in a table represents relationships for a unique entity, the table should have one column or a set of columns that provides a unique identifier for the rows of the table. This column (or set of columns) is called the *parent key* of the table. To ensure that the parent key does not contain duplicate values, you must create a unique index (see “Using Unique Indexes” on page 2-55) on the column or columns that constitute the parent key. Defining the parent key is thus called entity integrity, since it requires each entity to have a unique key.

In some cases, using a timestamp as part of the key can be helpful, for example when a table does not have a “natural” unique key or if arrival sequence is the key (see “Date, Time and Timestamp Data Types” on page 2-48).

Primary keys for some of the sample tables are:

Table	Key Column
Employee table	EMPNO
Department table	DEPTNO
Project table	PROJNO

Figure 28 shows part of the project table with the primary key column indicated.

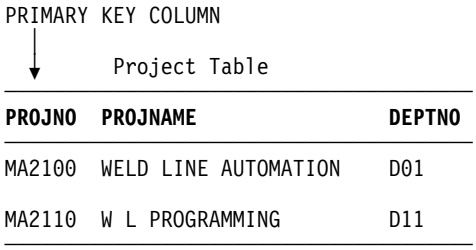


Figure 28. A Primary Key on a Table

Figure 29 shows a primary key containing more than one column; it is a *composite key*.

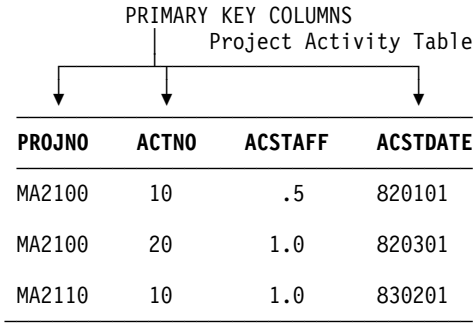


Figure 29. A Composite Primary Key. The three columns PROJNO, ACTNO, and ACSTDATE are all parts of the key.

Defining a Parent Key and a Unique Index

The information under this heading, up to “Defining a Foreign Key” on page 2-22 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

The primary key of a table, if there is one, uniquely identifies each occurrence of an entity about which the table contains information. The PRIMARY KEY clause identifies the column or columns of the primary key. Each column identified must be defined as NOT NULL.

| Another way to allow only unique values in a column is to create a table using the UNIQUE clause of the CREATE TABLE statement. Like the PRIMARY KEY clause, the specification of a UNIQUE clause prevents use of the table until you create an index to enforce the uniqueness of the key. For more information about the UNIQUE clause, see Chapter 6 of *SQL Reference*.

| A table that is to be a parent of dependent tables must have a primary or a unique key— the foreign keys of the dependent tables refer to it. Otherwise, a primary key is optional. Consider defining a primary key if each row of your table does pertain to a unique occurrence of some entity. If you define a primary key, an index must be created (the *primary index*) on the same set of columns, in the same order as those columns. If you are defining referential constraints for DB2 to enforce, read “Chapter 2-3. Maintaining Data Integrity” on page 2-19 before creating or altering any of the tables involved.

Further information about other clauses of the CREATE TABLE statement can be found in other sections of *Administration Guide* and in Chapter 6 of *SQL Reference*.

A table can have only one primary key. It obeys the same restrictions as do index keys:

- It can include no more than 64 columns
- No column can be named twice
- # The sum of the column length attributes cannot be greater than 255

You define a list of columns as the primary key of a table with the PRIMARY KEY clause in the CREATE TABLE statement.

To add a primary key to an existing table, use the PRIMARY KEY clause in an | ALTER TABLE statement. In this case, a unique index must already exist.

Incomplete Definition

If a table is created with a primary key, its *primary index* is the first unique index created on its primary key columns, with the same order of columns as the primary key columns. The columns of the primary index can be in either ascending or descending order. The table has an *incomplete definition* until the primary index is created. This incomplete definition status is recorded as an “I” in the STATUS column of SYSIBM.SYSTABLES. Use of a table with an incomplete definition is severely restricted: you can drop the table, create the primary index, and drop or | create other indexes; you cannot load the table, insert data, retrieve data, update | data, delete data, or create foreign keys that reference the primary key.

Because of these restrictions, you might plan to create the primary index soon after creating the table. For example, to create the primary index for the project activity table, issue:

```
CREATE UNIQUE INDEX XPROJAC1
  ON DSN8510.PROJACT (PROJNO, ACTNO, ACSTDATE);
```

Creating the primary index resets the “incomplete definition” status and its associated restrictions. But if you drop the primary index, it reverts to incomplete definition status; to reset it, you must create the primary index or alter the table to drop the primary key.

If the primary key is added later, with ALTER TABLE, then a unique index on the key columns must already exist. If there is more than one unique index on those columns, DB2 chooses one arbitrarily to be the primary index. See “Altering a Table for Referential Integrity” on page 2-130 for more information about altering an existing table.

Recommendations for Defining Primary Keys

Consider the following items when you plan for primary keys:

- The theoretical model of a relational database suggests that every table should have a primary key to uniquely identify the entities it describes. However, you must weigh that against the potential cost of index maintenance overhead. There is no DB2 requirement to define a primary key for tables with no dependents.
- Choose a primary key that cannot be updated. This enforces the good practice of having unique identifiers that remain the same for the lifetime of the entity occurrence.
- A primary key column should not have default values unless the primary key is a single TIMESTAMP column.
- Choose the minimum number of columns to ensure uniqueness of the primary key.
- An updatable view that is defined on a table with a primary key should include all columns of the key. Although this is necessary only if the view is used for inserts, the unique identification of rows can be useful if the view is used for updates, deletes, or selects.
- You can drop a primary key later if you change your database or application using SQL.

Defining a Foreign Key

The information under this heading, up to “Implications for SQL Statements” on page 2-25 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

You define a list of columns as a foreign key of a table with the FOREIGN KEY clause in the CREATE TABLE statement.

A foreign key can refer to either a unique or a primary key of the parent table. If the foreign key refers to a non-primary unique key, you must specify the column names of the key explicitly. If the column names of the key are not specified explicitly, the default is to refer to the column names of the primary key of the parent table.

The column names you specify identify the columns of the parent key. The privilege set must include the ALTER or the REFERENCES privilege on the columns of the parent key. A unique index must exist on the parent key columns of the parent table.

The Relationship Name

You can choose a constraint name (an identifier of up to 8 bytes) for the relationship defined by a foreign key, but you do not need to. If you do not choose a name, DB2 generates one from the name of the first column of the foreign key, in the same way that it generates the name of an implicitly created table space (as described under “Creating a Table Space Implicitly” on page 2-86). For example, the names of the relationships in which the employee to project activity table is a dependent would, by default, be recorded (in column RELNAME of SYSIBM.SYSFOREIGNKEYS) as 'EMPNO' and 'PROJNO'. In the following sample CREATE TABLE statement, they are given the constraint names REPAPA and REPAE.

```
CREATE TABLE DSN8510.EMPPROJACT
      (EMPNO     CHAR(6)          NOT NULL,
       PROJNO    CHAR(6)          NOT NULL,
       ACTNO     SMALLINT         NOT NULL,
       FOREIGN KEY REPAPA (PROJNO, ACTNO) REFERENCES DSN8510.PROJACT
               ON DELETE RESTRICT,
       FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8510.EMP
               ON DELETE RESTRICT         )
  IN DATABASE DSN8D51A;
```

Figure 30. Specifying Foreign Key Constraint Names

The name is used in error messages, queries to the catalog, and the DROP FOREIGN KEY statement. Hence, you might want to choose one if you were experimenting with your database design and had more than one foreign key beginning with the same column (when DB2 would otherwise generate the name).

Indexes on Foreign Keys

No index is required on a foreign key, but one is strongly recommended if rows of the parent table are often deleted. The validity of the delete, and its possible effect on the dependent table, can be checked through the index.

You can create an index on the columns of a foreign key in the same way you create one on any other set of columns. Most often it would not be a unique index. If you do create a unique index on a foreign key, it introduces an additional constraint on the values of the columns.

In order to let an index on the foreign key to be used on the dependent table for a delete operation on a parent table, the leading columns of the index on the foreign key must be identical to and in the same order as the columns in the foreign key.

A foreign key can also be the primary key; then the primary index is also a unique index on the foreign key. In that case, every row of the parent table has at most one dependent row. The dependent table might be used to hold information that pertains to only a few of the occurrences of the entity described by the parent table. For example, a dependent of the employee table might contain information that applies only to employees working in a different country.

For another example, the primary key could share columns of the foreign key if the first *n* columns of the foreign key are the same as the primary key's columns. Again, the primary index serves as an index on the foreign key. In the sample project activity table, the primary index (on PROJNO, ACTNO, ACSTDATE) serves as an index on the foreign key on PROJNO. It does not serve as an index on the foreign key on ACTNO, because ACTNO is not the first column of the index.

The FOREIGN KEY Clause in ALTER TABLE

You can add a foreign key to an existing table; in fact, that is sometimes the only way to proceed. To make a table self-referencing, you must add a foreign key after creating it.

When a foreign key is added to a populated table, the table space is put into *check pending* status. See “The Check Pending Status and Implications for CHECK DATA” on page 2-31 for more information.

Restrictions on Cycles of Dependent Tables

A *cycle* is a set of two or more tables that can be ordered so that each is a dependent of the one before it, and the first is a dependent of the last. Every table in the cycle is a descendent of itself. In the sample application, the employee and department tables are a cycle; each is a dependent of the other.

DB2 does not allow you to create a cycle in which a delete operation on a table involves that same table. Enforcing that principle creates rules about adding a foreign key to a table:

- In a cycle of two tables, neither delete rule can be CASCADE.
- In a cycle of more than two tables, two or more delete rules must not be CASCADE. For example, in a cycle with three tables, two of the delete rules must be other than CASCADE. This concept is illustrated in Figure 31.

On the other hand, a delete operation on a self-referencing table must involve the same table, and the delete rule there must be CASCADE or NO ACTION.

No rule, though, prevents you from creating a cycle in which all the delete rules are RESTRICT and none of the foreign keys allow nulls. But try not to do this because, when you have done it, there is no way to delete a row of any of the tables.

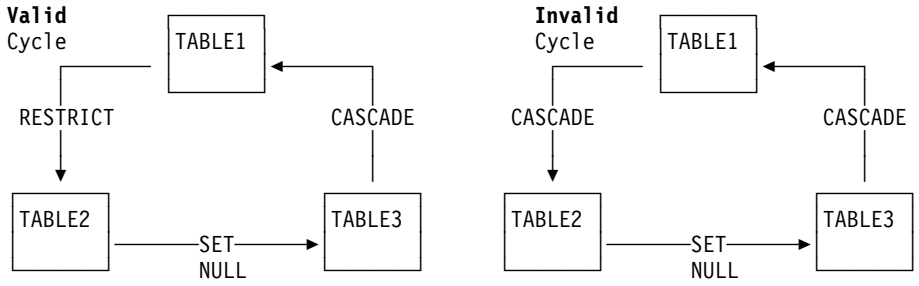


Figure 31. Valid and Invalid Delete Cycles. The left cycle is valid because two or more delete rules are not CASCADE. The cycle on the right is invalid because of the two cascading deletes.

Implications for SQL Statements

The information under this heading, up to “Implications for Utility Operations” on page 2-30 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

Suppose that relationships are defined on a set of tables, as in the sample application. The SQL operations INSERT, UPDATE, and DELETE must now comply with referential constraints.

The Basic Constraint: All of the changes combine to enforce this rule: a foreign key value in a dependent table must either match a parent key value in the corresponding parent table or be null.

If the foreign key consists of more than one column, its value is considered null if the value in any one of the columns is null.

Implications for INSERT

The basic constraint applies most clearly to an INSERT operation. For example, the project table has foreign keys on the department number (DEPTNO), referencing the department table, and the employee number (RESPEMP), referencing the employee table. Every row inserted in the table must have a value of RESPEMP that is either equal to some value of EMPNO in the employee table or is null. The row must also have a value of DEPTNO that is equal to some value of DEPTNO in the department table. (The null value is not allowed because DEPTNO in the project table is defined as NOT NULL.) An attempt to insert a row that would violate that rule results in an error code.

The department table is a *self-referencing table*. Column ADMRDEPT is a foreign key on column DEPTNO, both of which are in the department table. In general, when you insert a new row in the department table, a row for the department it reports to must already be there. Therefore, if the referential constraint is defined on the table when it is empty, the reporting department (A00) and the department it reports to must be the same in the first row you insert. This type of row is called a *self-refencing row* because the foreign key and its corresponding primary key are in the same row.

INSERT Rules

Again, the examples are based on the department and employee tables, considering only the relationship between the columns of department numbers.

For Parent Tables: You can insert a row at any time into a parent table without taking any action in the dependent table. For example, you can create a new department in the department table without making any change to the employee table.

For Dependent Tables: You cannot insert a row into a dependent table unless there is a row in the parent table with a parent key value equal to the foreign key value you want to insert. If a foreign key has a null value, it can be inserted into a dependent table, but then no logical connection exists.

Implications for UPDATE

If the unique key or primary key is a parent key, the referential constraints are effectively checked at the end of the operation.

For foreign key values, the restriction is essentially the same as for INSERT. You cannot change the value of a foreign key so that the dependent row no longer has a parent row, unless you make the key value null. For example, department numbers in the employee table depend on the department table; you can assign an employee to no department at all, but not to a department that does not exist. That is, you can change a value of WORKDEPT to null, but you cannot change it to a nonnull value that does not appear in the department table.

UPDATE Rules

Again, the examples are based on the department and employee tables.

For Parent Tables: You cannot change a parent key column of a row that has a dependent row. If you do, the dependent row no longer satisfies the referential constraint, so the operation is prohibited.

For Dependent Tables: You cannot change the value of a foreign key column in a dependent table unless the new value exists in the parent key of the parent table. For example, when an employee transfers from one department to another, the department number must change. The new value must be the number of an existing department, or be null.

Implementing the Consistency Rules

"Chapter 2-3. Maintaining Data Integrity" on page 2-19 describes the procedures to use to have DB2 automatically enforce referential constraints. It includes information on defining the referential constraints using SQL and the implications of those constraints.

Otherwise, you can implement the rules through application programs. Here, the designer of an application must describe the constraints that the application programs enforce. For example, you might simplify the task by limiting the number of application programs that perform insert and delete operations.

For some applications, a good alternative is not to enforce referential constraints but instead to accept the possibility of inconsistent data. For example, the sample employee table includes a job name and a code for education level for each employee. We could have set up master tables of allowable job names and education-level codes, and defined referential constraints to ensure consistency in job names and education-level codes in the employee table. However, we chose not to build these additional tables. Table check clauses would generally be a better choice for allowable job names and education level codes. We can use the flexibility and power of SQL to ensure consistency without needing additional table accesses.

Implications for DELETE

The effect of a delete operation on a parent table can become quite complex. There is no new effect on a row that is not a parent row; it is deleted. But the effect of an attempt to delete a row that has dependents varies according to the delete rules defined for the relationships in which it participates.

DELETE Rules

For ON DELETE RESTRICT, the attempt to delete a row with dependents fails with an SQL error return code. For example, you cannot delete a department from the department table as long as it is still responsible for some project, described by a dependent row in the project table. With ON DELETE RESTRICT, DB2 checks constraints before it performs each cascaded update and delete operation. Because this behavior would cause unpredictable results for a self-referencing constraint, you cannot specify ON DELETE RESTRICT for a self-referencing constraint.

For ON DELETE NO ACTION, the attempt to delete a row with dependents fails with an SQL error return code. With ON DELETE NO ACTION, DB2 checks constraints after it performs all cascaded update and delete operations. This means that for self-referencing constraints, the result of the delete operation is not dependent on the order in which DB2 processes rows in the table.

For ON DELETE SET NULL, the parent row is deleted, and in all its dependent rows the values of all columns of the foreign key that allow nulls are set to null. For example, you can delete an employee from the employee table even if the employee manages some department: the value of MGRNO in the department table is set to null. The change requires no privilege on the dependent table, only the DELETE privilege on the parent.

For ON DELETE CASCADE, the attempt to delete a parent row propagates to all its dependent rows in the dependent table, and from them to all their dependent rows, according to the delete rules encountered in each new relationship. If the delete rules are all CASCADE, then all descendent rows are deleted. For example, you can delete a department by deleting its row in the department table; this action also deletes the rows for all departments that report to it, all departments that report to them, and so on. Deleting the row for department A00 deletes every row in the table. The change requires no privilege on the descendent tables, only the DELETE privilege on the parent.

But if the cascading delete ever encounters a row that is dependent through a relationship with the rule RESTRICT, the operation is completely canceled. The parent row in the latest relationship cannot be deleted; hence, its parents cannot be deleted, nor their parents, and so on back to the original row. No rows are deleted.

Finally, encountering a relationship with the rule SET NULL merely ends one branch of the cascade. The foreign key in the final dependent table must have one or more columns that allow nulls. Because those columns cannot be part of the parent key of that table, setting those values to null has no effect on any of the table's dependents. For example, deleting a department from the department table sets WORKDEPT to null for every employee assigned to that department. But WORKDEPT is not part of the parent key of the employee table; therefore, the change has no effect on other rows.

DELETE with a Subquery: It is a basic rule of SQL that the result of an operation must not depend on the order in which rows of a table are accessed. That rule gives rise to the restriction that a subquery of a DELETE statement must not reference the same table from which rows are deleted.

For example, in the sample application some departments administer other departments; consider the following statement, which seems to delete every department that does not administer another one:

Invalid Statement

```
DELETE FROM DSN8510.DEPT THIS
  WHERE NOT EXISTS (SELECT * FROM DSN8510.DEPT
                      WHERE ADMRDEPT = THIS.DEPTNO);
```

If the statement can be executed, its result depends strongly on whether the row for any department is accessed before or after deleting the rows for the departments it administers; hence, it is prohibited.

Where there are referential constraints, the same rule extends to dependent tables. See Chapter 6 of *SQL Reference* for more information about using DELETE.

Operations on Self-Referencing Tables: The department table is self-referencing—every value of column ADMRDEPT is constrained to be a value of column DEPTNO. *Without* referential constraints, a single insert operation might insert two new rows for departments F01 and F11, where F01 administers F11; the order of inserting the rows makes no difference. *With* the constraints, the row for F01 has to be inserted before the row for F11. Because the result cannot be allowed to depend on the order of the rows, the operation is prohibited; a similar prohibition applies to delete operations.

If a self-referencing table is the object of an INSERT statement with a subquery, the subquery cannot return more than one row.

The examples that follow are based on the department and employee tables, considering only the relationship between the columns of department numbers.

For Parent Tables: Suppose, for example, that you delete the row about Department C01 from the department table. That deletion should affect the information in the employee table about Dolores Quintana and Heather Nicholls, who belong to that department. For any particular relationship, DB2 can enforce any one of the following delete rules:

- CASCADE

  When you delete a row of the parent table, any related rows in the dependent table are also deleted. This rule is useful when a row in the dependent table makes no sense without a row in the parent table.

  For example, a row in a sales table could represent a quantity of items sold and refer to a table of purchase orders; if the purchase order is deleted, all the item quantities listed in it should be deleted. But the rule would be inappropriate for the department-employee relationship: when you dissolve a department, you do not want to throw away the records of employees currently assigned to it.

  CASCADE is a very powerful rule. Unlike RESTRICT and SET NULL, CASCADE can potentially trickle down many levels of descendents if those descendents also use the CASCADE delete rule. Use this rule with caution.

- RESTRICT or NO ACTION

  You cannot delete any rows of the parent table that have dependent rows. In the department-employee relationship, using RESTRICT or NO ACTION requires that you reassign every employee in a department before you can delete the department. The only difference between NO ACTION and RESTRICT is *when* the referential constraint is enforced. RESTRICT enforces the rule immediately and NO ACTION enforces the rule at the end of the statement.

- SET NULL

  When you delete a row of a parent table, the corresponding values of the foreign key in any dependent rows are set to null only if the column allows a NULL value; columns that do not allow NULL values remain unchanged. This rule is used in the department-employee relationship; when you delete a department record, the WORKDEPT column of related rows in the employee table is set to null, indicating that the employees are not assigned to a department.

For Dependent Tables: You can at any time delete rows from a dependent table without taking any action on the parent table. For example, in the department-employee relationship, an employee could retire and have his row deleted from the employee table with no effect on the department table. (Ignore, for the moment, the reverse relationship of employee-department, in which the department manager ID is a foreign key referring to the primary key of the employee table. If a manager retires, there is an effect on the department table.)

Implications for DROP

Dropping a table is not equivalent to deleting all its rows, and the operation does not propagate according to referential delete rules. Instead, when you drop a table you drop all the relationships in which the table participates, either as a parent or a dependent. For example, dropping the activity table from the sample set would not affect the rows of the project activity table at all. But that table would no longer have a foreign key on the ACTNO column, and DB2 would not check the values in that column when rows were inserted or updated. In such circumstances, remember that application programs might depend on the existence of a parent table, and use DROP with care.

Performance Implications

Referential constraints can have a significant impact on the performance of DELETE operations. That is especially true for a cascading delete, in which an operation that originally accesses a single row could propagate to hundreds of rows in dozens of tables.

In many cases, the impact can be reduced by creating indexes on the foreign keys. An index is not required on a foreign key, but without one a deletion of a row from the parent table requires a scan of the entire dependent table—possibly multiple scans of many dependent tables!

In some cases, the performance impact can be further reduced by the placement of tables in table spaces. If rows of table T are often deleted, consider placing in the same table space all the dependents of T and all the tables to which deletes from T cascade. But if a table has no index on the foreign key, consider placing it in its own table space, or in a segmented table space, to avoid scanning extraneous rows when deletes cascade to it.

Concurrent Operations: A delete operation on a parent table must acquire locks on the dependent tables, or at least on their indexes. That can only make those tables less readily available for concurrent use.

Locks during Bind: Table spaces and index spaces that are required only for enforcing referential constraints are not affected by the ACQUIRE(ALLOCATE) option of the BIND PLAN command (there is no ACQUIRE(ALLOCATE) option for

BIND PACKAGE). The table spaces and indexes spaces are acquired only when used, and the time needed for that operation is also a performance consideration. With the RELEASE(DEALLOCATE) option, they are kept open, like all other table and index spaces, until the plan terminates.

Implications for Utility Operations

Enforcing referential constraints during utility operations significantly changes the effects of several utilities. Also, there are operations that aid in checking constraints and correcting any errors that are discovered.

Implications for LOAD

For tables with primary keys: LOAD does not load a table with an incomplete definition. If the table has a parent key, the unique index on that key must exist. If you try to load any table that has an incomplete definition, the LOAD job terminates.

For tables with foreign keys: By default, LOAD enforces referential constraints. That is, it does not load a row with a foreign key value that does not match some parent key value of a parent table. Encountering such a row produces a message and the row can be written to a discard data set.

Concurrency considerations for referential constraints: LOAD requires access to the primary indexes on the parent tables of any tables loaded. For simple, segmented, and partitioned table spaces, it drains all writers from the parent tables' primary indexes. Other users cannot make changes to the parent tables that result in an update to their own primary indexes. Concurrent inserts and deletes on the parent tables are blocked, but updates are allowed for columns not defined as part of the primary index.

For cycles of dependent tables: For a cycle of dependent tables, referential constraints must sometimes be suspended; so there is an ENFORCE(NO) option. For example, MGRNO in the department table is a foreign key that references EMPNO in the employee table; WORKDEPT in the employee table is a foreign key that references DEPTNO in the department table. If you want to add a new department and its employees, using LOAD(RESUME), you can consider these methods:

- If both tables are in the same table space, load the new records to both tables in the same job, while enforcing the constraints. All records are loaded before the constraints are checked; therefore, if all was prepared correctly, the new department record and the new manager's employee record are in place before checking begins. But this option does not apply to the sample application, because the two tables are in different table spaces.
- If one of the tables allows null values in its foreign key, load it with nulls there, while enforcing referential constraints. (The null value of the foreign key is always valid.) Then load the second table, again enforcing referential constraints. Finally, update the null values in the first table. This option is reasonable for the sample application. Both tables allow null values, and it is a simple matter to choose the department table as the first table and load (or insert) a single new department with a null value of MGRNO. However, the method is more difficult when many rows are added in each table; in particular, the final update can become quite complex.

- Load new records to one table without enforcing referential constraints (use ENFORCE(NO)). Then load the second table, enforcing the constraints. Finally, run the CHECK DATA utility to verify that records loaded with constraint checking turned off do not violate any referential constraints. Running CHECK DATA removes the check pending restriction placed on the first table space by the load that was not enforced.

The Check Pending Status and Implications for CHECK DATA

If a table is loaded without enforcing referential constraints on its foreign keys, it can then contain data that violates the constraints. Using that data could threaten the referential integrity of an entire set of related tables. So the table space containing the table is immediately placed in a special status called *check pending*. For partitioned table spaces, only the newly loaded partitions are placed in check pending. Until the status is reset, none of the tables in the table space can be used for SQL SELECT, INSERT, UPDATE, or DELETE operations. The table space cannot be the object of a COPY, REORG, or QUIESCE utility job. And INSERT, UPDATE, and DELETE operations on tables in other table spaces do not execute if those operations, through referential constraints, involve a table in the table space that is in the check pending status.

Other operations that also can cause a table space to be placed in check pending status are:

- Defining a referential constraint on a populated table, using ALTER TABLE. (A table in a nonsegmented table space is considered populated if it has ever contained records, even if all were deleted.)
- Interrupting a LOAD job before the checking of referential constraints is complete.
- Replacing the data in a parent table space, using LOAD REPLACE. The table spaces containing all the dependent tables of the parent are placed in check pending status immediately after the parent table space has been reset. This is the point at which the constraint violations are introduced.

Certain CHECK DATA and RECOVER operations, described below, can also place table spaces in the check pending status.

The Scope of Check Pending Status

CHECK DATA need not always examine an entire table space; a scope is recorded with each check pending status, and only rows within the scope must be checked. For example, if rows are loaded to a table with RESUME(YES) and ENFORCE(NO), only the new rows need checking.

Resetting the Check Pending Status

The CHECK DATA utility checks whether the tables of a table space violate referential constraints and, optionally, deletes any invalid rows. If no rows violate the constraints, or if invalid rows are deleted, the utility resets the check pending status of the table space. The deleted rows can be copied to *exception tables*, where they can be examined and corrected, and from which they can be reinserted in the original tables.

Creating Exception Tables

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

Table 5. Exception Tables

Column	Description	Required	Data Type and Length	NULL Attribute
1 to n	Corresponds to columns in the table being checked and are used to hold data from rows in the table being checked that violate referential constraints.	Yes	The same as the corresponding columns in the table being checked.	The same as the corresponding columns in the table being checked.
n+1	Identifies the RIDs of the invalid rows of the table being checked.	No	CHAR(4) or CHAR(5)	Anything
n+2	Starting time of the CHECK utility	No	TIMESTAMP	Anything
≥ *n+2*	Any additional columns are not used by the CHECK utility	No	Anything	Anything

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

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

- The exception tables should not have any unique indexes or referential constraints that could cause errors when CHECK DATA inserts rows in them.
- You can create a new exception table prior to running the CHECK DATA utility or use an existing exception table. The exception table can contain rows from multiple invocations of the CHECK DATA utility.
- If column *n+2* is of type TIMESTAMP, CHECK DATA records the starting time. Otherwise, it does not use this column.
- The RID column used by the CHECK utility must be 5 bytes for LARGE table spaces but it can be 4 or 5 bytes for table spaces that are not defined with LARGE.
- The user must have DELETE authorization on the dependent table being checked.
- The user must have INSERT authorization on the exception table.
- Exception table column names can be given any name.
- Any change to the structure of the dependent table (such as a column being dropped or added) is not automatically reflected in the exception table. You must make that change to the exception table yourself.

General-use Programming Interface

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

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

ALTER TABLE EPROJACT
   ADD RID CHAR(5);

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

The first statement requires the SELECT privilege on table DSN8510.PROJACT as well as the privileges usually required to create a table.

End of General-use Programming Interface

Table EPROJACT has the same structure as table DSN8510.PROJACT, but with two extra columns.

- Its first five columns mimic the columns of the project activity table; they have exactly the same names and descriptions. Although the column names are the same, they do not have to be. However, the rest of the column attributes for the initial columns must be same as those of the table being checked.
- The next column, added by ALTER TABLE, is optional; CHECK DATA uses it as an identifier. The name “RID” is an arbitrary choice—if the table already has a column with that name, you have to use something else. But the description, CHAR(4) or CHAR(5), is required.
- The final timestamp column is also optional and useful. If the timestamp column is defined, a row identifier (RID) column must exist and precede this column. You might define a permanent exception table for each table subject to referential constraints. You can define it once and use it as you wish to hold invalid rows detected by CHECK DATA. The TIME column allows you to identify rows added by the most recent run.

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

General-use Programming Interface

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

End of General-use Programming Interface

Other Ways to Reset Check Pending Status

The purpose of the check pending status is to encourage the use of CHECK DATA, but the status can also be reset by any of the following operations:

- Dropping tables, so the table space no longer contains invalid rows.
- Replacing the data in the table space, using LOAD REPLACE and enforcing the referential constraints.
- Recovering all members of an inter-related set of tables to a quiesce point, when no constraints were violated.
- Dropping all foreign keys in the table using ALTER TABLE. The check pending status is reset when there are no more foreign keys defined in the table.

Implications for COPY, QUIESCE, RECOVER, and REPORT

All of the following utilities are concerned with recovering or preparing to recover data. “Chapter 4-6. Backing Up and Recovering Databases” on page 4-123 describes the operations in detail; here they are outlined as considerations for relational database design.

Where there are referential constraints, recovery requires careful attention, for it is possible to recover one of a set of related tables to a state in which it is inconsistent with the others. DB2 guards against such an event, but does not absolutely prevent it.

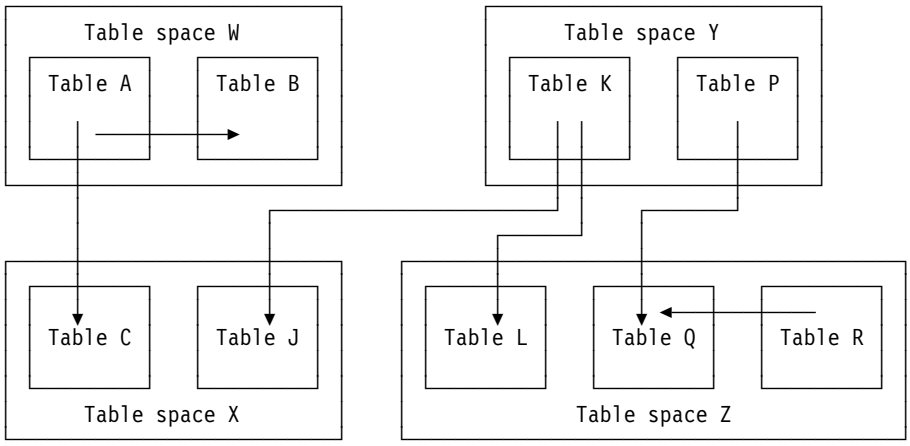


Figure 32. Recovery and Referential Structures. Recovering Tables in Referential Structures

When you define referential constraints, you create referential structures, made up of all objects connected by referential relationships. In Figure 32, tables A, B and C form a referential structure, as do tables J, K and L, and tables P, Q and R. If you recover table A to a prior point, you must recover tables B and C to the same point to preserve the integrity of the structure. Because COPY and RECOVER operate on table spaces, you recover table space W and table space X, which also contains table J. To preserve the integrity of the J-K-L referential structure, you must recover table spaces Y and Z. Note that this also recovers all of the tables in the P-Q-R structure.

The COPY Utility

COPY cooperates in enforcing referential constraints by declining to copy a table space in check pending status. It is probably not necessary to copy every table space in a set at every quiesce point. The RECOVER utility can use incremental image copies and log records.

The QUIESCE Utility

If there are many table spaces in a set, all being updated concurrently by many users, you might need a special technique to establish a point in time at which the entire set is known to be consistent, no more work is going on, and the set can be copied. The technique is to start the table spaces for read-only access, run the QUIESCE utility (to wait for the clean point and record it), copy all the table spaces, then start the table spaces for read-write access. See *Utility Guide and Reference* for more details about QUIESCE.

The RECOVER Utility

Use the DB2 RECOVER utility to recover a list of table spaces to the point in time recorded by QUIESCE. The recovery is swiftest, of course, if all the table spaces have been copied at that point. For instructions, see “Using RECOVER to Restore Data to a Previous Point in Time” on page 4-147.

Recovery to a prior time can either set or reset the check pending status. RECOVER sets the status on if:

- The list of table spaces does not make up a complete table space set. In that case, the status is set on for every dependent space that is recovered, and for every space anywhere that is a dependent of a recovered space.
- The list made up a complete table space set, but the prior time is not a quiesce point nor an image copy made with SHRLEVEL REFERENCE for some table space recovered in the set.
- All table spaces are recovered to a quiesce point or to an image copy made with SHRLEVEL REFERENCE, but a referential constraint or a check constraint was defined on some table in one of the table spaces after that point.

If none of these conditions occurs, then RECOVER resets the check pending status of any table space in the list.

In summary, creating a referential structure codifies an association among the members of a table space set. In utility operations, especially loading and recovering, all tables in the set must be considered together. But, even without referential constraints, the association exists; it is implicit in the relationships among the entities that the data describes. The difference is that, with the constraints defined, you cannot choose to simplify operations by accepting inconsistencies in the data.

The REPORT Utility

In most cases, you want only a single referential structure in each table space set. But there might be some question whether the database design was implemented as intended. Perhaps users have been allowed to create tables in several different table spaces and to define relationships among them. When in doubt, use the REPORT utility to list the names of all the table spaces in a set, given any one of them, using:

```
REPORT TABLESPACESET TABLESPACE tsname
```

Defining Table Check Constraints

The information under this heading, up to “Column Specifications” on page 2-41 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

When designing your tables, consider whether you need table check constraints. Table check constraints designate the values that specific columns of a base table can contain, providing you a method of controlling the integrity of data entered into tables. You can create tables with table check constraints using the CREATE TABLE statement, or add the constraints with the ALTER TABLE statement. However, if the *check integrity* is compromised, or cannot be guaranteed for a table, the table space or partition that contains the table is placed in a check pending state. *Check integrity* is the condition that exists when each row of a table conforms to the check constraints defined on that table.

For example, you might want to make sure that no salary can be below 15,000 dollars:

```
CREATE TABLE EMPSAL
(ID          INTEGER      NOT NULL,
 SALARY      INTEGER      CHECK (SALARY >= 15000).
```

Figure 33. Creating a Simple Table Check Constraint

Using table check constraints makes your programming task easier, because you do not have to enforce those constraints within application programs or with a validation routine. Define table check constraints on one or more columns in a table when that table is created or altered.

Constraint Considerations

The syntax of a table check constraint is checked when the constraint is defined, but the meaning is not checked. The following examples show mistakes that are not caught. Column C1 is defined as INTEGER NOT NULL.

Allowable but mistaken check constraints:

- A self-contradictory check constraint:

  ```
  CHECK (C1 > 5 AND C1 < 2)
  ```

- Two check constraints that contradict each other:

  ```
  CHECK (C1 > 5)
  CHECK (C1 < 2)
  ```

- Two check constraints, one of which is redundant:

  ```
  CHECK (C1 >  0)
  CHECK (C1 >= 1)
  ```

- A check constraint that contradicts the column definition:

  ```
  CHECK (C1 IS NULL)
  ```

- A check constraint that repeats the column definition:

  ```
  CHECK (C1 IS NOT NULL)
  ```

A table check constraint is not checked for consistency with other types of constraints. For example, a column in a dependent table can have a referential constraint with a delete rule of SET NULL. You can also define a check constraint that prohibits nulls in the column. As a result, an attempt to delete a parent row fails, because setting the dependent row to null violates the check constraint.

Similarly, a table check constraint is not checked for consistency with a validation routine, which is applied to a table before a check constraint. If the routine requires a column to be greater than or equal to 10 and a check constraint requires the same column to be less than 10, table inserts are not possible. Plans and packages do not need to be rebound after table check constraints are defined on or removed from a table.

When Table Check Constraints Are Enforced

After table check constraints are defined on a table, any change must satisfy those constraints if it is made by:

- The LOAD utility with the option ENFORCE CONSTRAINT
- An SQL INSERT statement
- An SQL UPDATE statement

A row satisfies a check constraint if its condition evaluates either to true or to unknown. A condition can evaluate to unknown for a row if one of the columns named contains the null value for that row.

Any constraint defined on columns of a base table applies to the views defined on that base table.

When you use ALTER TABLE to add a table check constraint to already populated tables, the enforcement of the check constraint is determined by the value of the CURRENT RULES special register as follows:

- 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 there might be rows in the table that violate the check constraint, the table is placed in check pending status.

How Table Check Constraints Set Check Pending Status

Maintaining check integrity requires enforcing check constraints on data in a table. When check integrity is compromised or cannot be guaranteed, the table space or partition that contains the table is placed in check pending status. The definition of that status includes violations of table check constraints as well as referential constraints.

Table check violations place a table space or partition in check pending status when:

- A table check constraint is defined on a populated table using the ALTER TABLE statement and the value of the CURRENT RULES special register is DB2.
- The LOAD utility is run with CONSTRAINTS NO, and table check constraints are defined on the table.

- CHECK DATA is run on a table that contains violations of table check constraints.
- A point-in-time RECOVER TABLESPACE introduces violations of table check constraints.

Chapter 2-4. Designing Columns

This chapter describes how to design columns for tables, including column specifications and column names. It also describes column values, date, time, and timestamp data types, and some implementation information.

Choosing Columns

You implement your logical database design primarily by choosing the columns that make up each table. There is almost always some conflict between the theoretical design and the most practical implementation of the design. Some sources of that conflict are described under the headings “Consider Denormalizing Your Tables for Performance” on page 2-16 and “Considerations for Record Size.”

Considerations for Record Size

In DB2, a record is the storage representation of a row. Records are stored within 4 KB or 32 KB pages and a single record cannot occupy more than one page. You cannot create a table with a maximum record size greater than the page size. There is no other absolute limit, but to ignore record size in favor of implementing a good theoretical design might waste storage.

Record Length—Fixed or Varying

Most DB2 data types describe data of fixed length. Some data types, however, allow data of varying-length. In a table whose columns all have fixed-length data types, all rows (thus all records) are the same size. Otherwise, the size of records might vary. Record size might also vary if the table uses an edit routine or data compression.

Fixed-length records are preferable to varying-length records, because DB2 processing is most efficient for fixed-length records. A fixed-length record never has to be moved from the page on which it is first stored. Varying-length records, however, can be updated to a length that no longer fits on the original page; in that case, the record is moved to another page. When the record is accessed, there is an additional page reference. Therefore, use varying-length columns with care.

There is a performance advantage in placing varying-length columns after the fixed-length columns of the table. For more information, see “String Data Types” on page 2-45.

When you are using varying-length columns, there are two considerations: retrieval performance and update performance. For the best retrieval performance, place varying-length columns at the end of a row. For the best update performance, place varying-length columns at the end of a row. If you use both retrieval and update operations, place the columns you want to update at the end, followed by the read-only varying-length columns.

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.

Record Lengths and Pages

In addition to the bytes of actual data in the row, each record has:

- A 6-byte prefix
- One additional byte for each column that might contain null values
- Two additional bytes for each varying-length column.

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, so that if two records are to fit in a page, each cannot be longer than 1934 bytes (approximately $0.95 \times 4074 \times 0.5$). Furthermore, the record length is limited by the page size of the table space in which the table is defined. If the table space is 4 KB, then the record length of each row cannot be greater than 4056 bytes. Because there is an 8-byte overhead for each row, the sum of column lengths cannot be greater than 4048 bytes (4056 minus the 8-byte overhead for a record).

DB2 provides a 32KB page size to allow for long records. You can improve performance by using 32KB pages for record lengths that are longer than 2028 bytes, as in some text-processing or image-processing applications.

Designs That Waste Space

Space is wasted in a table space that only contains records slightly longer than half a page,because only one record can fit in a page. If you can reduce the record length to just under half a page, you need only half as many pages. Similar considerations apply to records that are just over (and could be reduced to just under) a third of a page, a quarter of a page, and so on.

Provide Column Definitions for All Tables

To define a column in a DB2 table:

1. Choose a name for the column

   Each column in a table must have a name that is unique within the table. Selecting column names is described in detail in “Column Specifications” on page 2-41.

2. Tell what kind of data is valid for the column

   The data type of a column indicates the length of the values in the column and the kind of data that is valid for the column. Data types are described in “Specifying Data Types” on page 2-44.

3. Tell which columns might need default values

   Some columns cannot have meaningful values in all rows because:

   - A value of the column is not applicable to the row.

For example, a column containing an employee's middle initial is not applicable to an employee who has no middle initial.

- A value is applicable, but the value is not known at this time.

  As an example, the MGRNO column might not contain a valid manager number because the previous manager of the department has been transferred and a new manager has not been appointed yet.

In both situations, you can choose between allowing a null value (a special value indicating that the column value is unknown or inapplicable) or allowing a nonnull default value to be assigned by DB2 or by the application.

Null values and default values are described in detail in “Null Values” on page 2-42.

Column Specifications

The information under this heading, up to “Chapter 2-5. Designing Indexes” on page 2-51 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

Tables without foreign keys and parent tables can have up to 750 columns. Tables with foreign keys can have up to 749 columns.

A column contains values of the same type. You can think of it as a field in a record. In the sample employee table, the HIREDATE column contains all the hire dates for all employees represented by EMPNO. You cannot redefine or overlap columns and, after you have implemented the design of your tables, you usually cannot change a column definition without disrupting applications. Therefore, consider carefully the decisions you make about column definitions. (However, you can add columns to an existing table; for instructions, see “Altering Tables” on page 2-128.)

For each column, you must specify a name and a data type. If you specify a string type, you also have to specify a length (of values in the column, not the number of values) and whether null values are permitted in the column.

You might want to execute a user-written exit routine whenever an application program enters data in the column or retrieves it. If you do, you must specify that also. This type of routine is called a *field procedure*; one can be used, for example, to alter the sorting sequence of values entered in the column. How you can use a field procedure is described under “Field Procedures” on page X-57.

Column Names

Column names must be unique within a table, but you can use the same column name in different tables. The maximum length of a column name is 18 bytes.

Column Labels

Frequently, a column name does not adequately describe the contents of a given column; you might want a more descriptive column heading to appear in an interactive display. The SQL LABEL ON statement provides a way for you to define expanded column headings that can appear in addition to, or instead of, column names in SPUFI output or in the SQL descriptor area (SQLDA) in application programs. For a description of this statement, see Chapter 6 of *SQL Reference*.

When you want to use the alternate column headings or labels on SPUFI output, specify LABELS in the COLUMN HEADINGS option on the SPUFI Default Panel.

Null Values

As described under “Provide Column Definitions for All Tables” on page 2-40, some columns cannot have a meaningful value in every row.

DB2 uses a special value indicator, called the *null value*, to stand for an unknown or missing value. A null value is a value and not a zero value, a blank, or an empty string. It is a special value interpreted by DB2 to mean that no data has been supplied.

If you do not specify otherwise, any column you define can contain null values, and rows can be created in the table without providing a value for the column.

NOT NULL disallows null values in the column. If you use NOT NULL and do not use some form of the DEFAULT clause, you must provide a nonnull value for that column whenever you insert data into the table.

Default Values

When a row is inserted or loaded and no value is specified for the column, the default value you specified with the DEFAULT clauses in CREATE TABLE or ALTER TABLE is used. Defaults can be either DB2-defined or user-defined as follows.

DB2-defined Defaults: If you specified DEFAULT without a value, the data type of the column determines the DB2-defined default as follows:

Table 6. DB2-defined default values for data types.

For columns of...	Data types	Default
Numbers	SMALLINT, INTEGER, DECIMAL, or FLOAT	0
Fixed-length strings	CHAR or GRAPHIC	Blanks
Varying-length strings	VARCHAR, LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC	Empty string
Dates	DATE	CURRENT DATE
Times	TIME	CURRENT TIME
Timestamps	TIMESTAMP	CURRENT TIMESTAMP

Note: Synonyms exist for the various data types.

User-defined Defaults: You can specify a particular default, such as

```
DEFAULT 'N/A'
```

The default value must agree with the data type of the column. You can specify one of the following forms:

- A constant
- The value of the USER special register at the time that DB2 inserts data into a row or updates the row.

- The SQL authorization ID (CURRENT SQLID) of the process at the time that DB2 inserts data into a row or updates the row.
- Null (provided the column allows nulls)

If you execute the ALTER TABLE statement to add a column to a table, and you define that column with DEFAULT USER or DEFAULT CURRENT SQLID, DB2 returns the value of USER or CURRENT SQLID for the new column when you select rows that existed before you executed ALTER TABLE.

For example, if you want a record of who inserted any row of a table, define the table with two additional columns:

```
PRIMARY_ID      CHAR(8)     WITH DEFAULT USER,
SQL_ID          CHAR(8)     WITH DEFAULT CURRENT SQLID,
```

You can allow updates and inserts to the table only through a view that omits those columns. Then, the primary authorization ID and the SQL ID of the process are added by default.

If you add a column to an existing table, and you read from it before you have added data to it, the values you retrieve are provided by default. The default values for retrieval are the same as the defaults for insert, except in the cases of columns with a data type of DATE, TIME, or TIMESTAMP. The retrieval defaults for the exceptional columns are:

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

Reasons for Using Nulls

Suppose you want to find out the average salary earned in a department. It is not necessary that the salary column always contain a meaningful value, so you can choose between:

- Allowing null values for the SALARY column
- Using a nonnull default value

By allowing null values, you can formulate the query easily, and DB2 provides the average of all known or recorded salaries. The calculation does not include the rows containing null values. In the second case, you get a misleading answer unless you know the nonnull default value for unknown salaries and formulate your query accordingly.

In Figure 34 on page 2-44, in the example using a nonnull default value, the average salary for department E21 is calculated on the basis of three employees, although salaries are given for only two. In the example using null values, only those employees with actual salary data are included in the calculation.

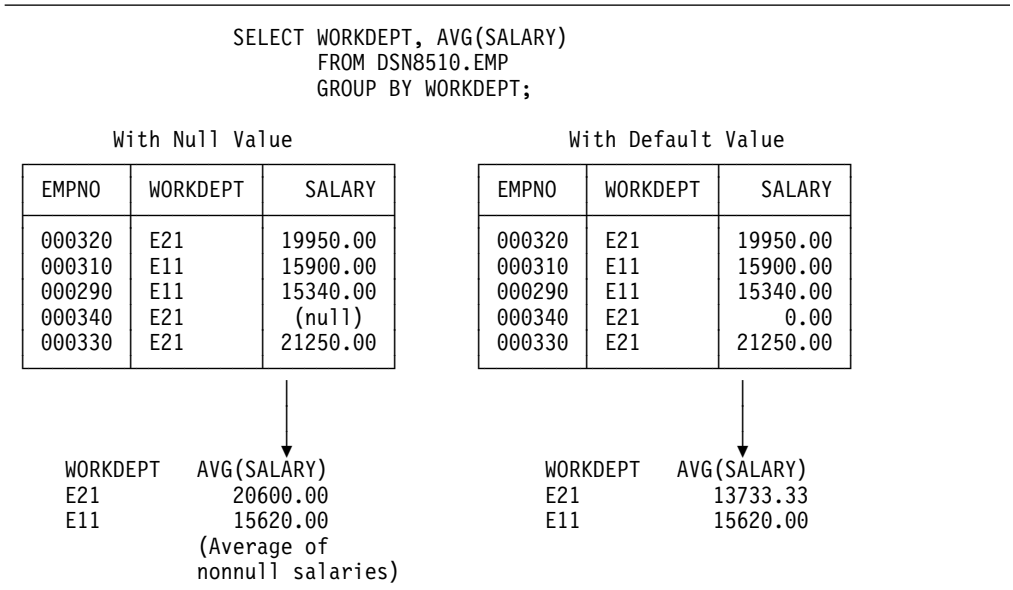


Figure 34. When Nulls Are Preferable to Default Values

Reasons for Using Nonnull Default Values

Before you decide whether to allow nulls for unknown values in a column, you should be aware of how nulls can affect the result of a query.

- Nulls in application programs

  Nulls do not satisfy any condition in an SQL statement other than the special IS NULL predicate. Null values do not behave like other values. For instance, if you ask DB2 to determine whether a null value is larger or smaller than a given known value, you get an answer of UNKNOWN for both comparisons.

  A default acts in a way familiar to programmers.

- Nulls in a join operation.

  Nulls need special handling. If you perform a join operation using a column that is allowed to contain null values, you should consider using an outer join. See Section 2 of *Application Programming and SQL Guide* for more information.

Nulls with Field Procedures: If you allow nulls in a column with a field procedure, that routine is not executed when you access a null value; rather, DB2 returns the null value. Field procedures can only be specified for short string columns that do not have a nonnull default value.

Specifying Data Types

You must give a data type for each column of a DB2 table. It tells the type of data the column will contain and the length of the data field.

The first thing you must decide when defining a column is what kind of data the column will contain—string, numeric, or date/time. The decision is often obvious because only a string column can contain letters or special characters. If the data consists solely of digits, however, you have to decide whether to specify it as string or numeric data. And if the values represent dates, times, or timestamps, you will want to consider the data types DATE, TIME, and TIMESTAMP.

Choosing String or Numeric Data Types

For numeric data, use numeric rather than string columns; they require less space, and DB2 verifies that the data has the assigned type. For example, if numbers are represented as strings, when DB2 calculates a range (as for BETWEEN, greater than, and less than) it assumes that its values include all combinations of alphanumeric characters. Consider defining numeric identifiers as numeric rather than string, and dates or times as DATE or TIME.

String Data Types

The data types for strings are described in Table 7. In each case the length specification, *(n)*, can be omitted (unless otherwise noted in the table); its default value is 1.

Table 7. String Data Types

Data Type	Denotes a column of...
CHARACTER(*n*)	Fixed-length character strings with a length of *n* bytes. *n* must be greater than 0 and not greater than 255.
VARCHAR(*n*)	Varying-length character strings with a maximum length of *n* bytes. The length *n* is required. *n* must be greater than 0 and less than a number that depends on the page size of the table space. If *n* is greater than 254, certain restrictions apply to the use of the columns in SQL statements. Appendix A of *SQL Reference* lists the upper limits on the value of *n*.
LONG VARCHAR	Varying-length character strings with a maximum length calculated by DB2. See the CREATE TABLE statement in Chapter 6 of *SQL Reference* to see how the length is calculated.
GRAPHIC(*n*)	Fixed-length graphic strings containing *n* double-byte characters. *n* must be greater than 0 and less than 128.
VARGRAPHIC(*n*)	Varying-length graphic strings. The maximum length, *n*, must be greater than 0 and less than a number that depends on the page size of the table space. The length *n* is required. If *n* is greater than 127, certain restrictions apply to the use of the column in SQL statements. Appendix A of *SQL Reference* lists the upper limits on the value of *n*.
LONG VARGRAPHIC	Varying-length graphic strings with a maximum length calculated by DB2.

String Subtypes: Subtypes can be important if another DBMS accesses your table through the distributed data facility and also uses a different character encoding scheme from your own. Subtypes apply only to CHAR, VARCHAR, and LONG VARCHAR data types. Table 8 describes the possible subtypes.

Table 8 (Page 1 of 2). String Subtypes

Subtype	Denotes that the data...
BIT	Does not represent characters, and is not converted as though it represents characters.

Table 8 (Page 2 of 2). String Subtypes

Subtype	Denotes that the data...
SBCS	Represents single-byte characters. The coded character set is specified by the single-byte character set (SBCS) coded character set identifier (CCSID) identified for this DB2 subsystem during installation. If required, values placed into this column or taken from this column are converted from one CCSID to another CCSID. This can occur, for example, if data with a CCSID different than the DB2 SBCS CCSID is received from a remote location for insertion into this column. This subtype is the default if the MIXED DATA option was set to NO during installation.
MIXED	Represents either SBCS characters or double-byte character set (DBCS) characters, or a combination of both. The SBCS and DBCS coded character sets are implied by the MIXED CCSID identified for this DB2 subsystem during installation. If required, values placed into this column or taken from this column are converted from one coded character set to another. This subtype is the default if the MIXED DATA option was set to YES during installation.

If bit data might be converted when accessed by a remote DBMS, use the clause FOR BIT DATA in the column description in the CREATE TABLE statement. For further information on character conversion, see *Installation Guide*.

Choosing EBCDIC or ASCII Data: If you require data to be in the ASCII sorting order, you can store and manipulate character data as ASCII. To do this, specify the ASCII value of the CCSID option on the CREATE TABLE, CREATE TABLESPACE, or CREATE DATABASE statement. See Chapter 6 of *SQL Reference* for details. Before you decide to use the ASCII encoding scheme for your data, consider the following restrictions:

- You must declare an entire table with a single encoding scheme. Some columns in the table cannot be ASCII when others are EBCDIC.
- If an SQL statement accesses more than one table, all tables must use the same encoding scheme.
- All tables in a referential structure must use the same encoding scheme.

Choosing CHAR or VARCHAR: VARCHAR saves DASD space, but costs a 2-byte overhead for each value and the additional processing required for varying-length records. Thus, CHAR is preferable to VARCHAR, unless the space saved by the use of VARCHAR is significant. The savings are not significant if the maximum length is small or the lengths of the values do not have a significant variation. In general, do not define a column as VARCHAR(*n*) unless *n* is at least 18. (Consider, also, using data compression if your main concern is DASD savings. See “Compressing Data in a Table Space or Partition” on page 2-63 for more information.)

If you use VARCHAR, do not specify a maximum length greater than necessary. Though VARCHAR saves space in a table space, it does not save space in an index (type 1 and type 2) because index records are padded with blanks to the maximum length. Note particularly the restrictions on columns of strings longer than

255 bytes; for example, they cannot be indexed. These restrictions are listed in Chapter 3 of *SQL Reference*.

Do not use LONG VARCHAR unless you really want the maximum record length to be as large as possible. Also, you cannot use ALTER to add a column to a table that already has a LONG column.

In most cases, the content of the data intended for a column dictates the data type you choose. For example, the data type selected for the department name (DEPTNAME) of the department table is VARCHAR(36). Because department names normally vary considerably in length, the choice of a varying-length data type seems appropriate. Choosing a data type of CHAR(36), for example, would result in much wasted space. All department names, regardless of length, would be assigned the same amount of space (36 bytes). The choice of a data type of CHAR(6) for the employee number would appear to be an obvious choice, because all values are fixed-length (6 bytes).

There is a performance advantage in placing VARCHAR columns after the fixed-length columns of the table.

The foregoing considerations about CHAR, VARCHAR, and LONG VARCHAR columns apply in the same way to GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC columns. The one exception is that the length (*n*) of a GRAPHIC or VARGRAPHIC column is given as a number of double-byte characters; hence, the length in bytes is twice *n*.

Choosing GRAPHIC or Mixed Data: Columns containing DBCS characters can be defined as graphic string columns (GRAPHIC, VARGRAPHIC, or LONG GRAPHIC) or as mixed data character string columns (CHAR, VARCHAR, or LONG VARCHAR with MIXED DATA implicitly or explicitly specified).

DBCS characters include Kanji characters and non-Kanji characters. The non-Kanji characters include double-byte representations for those characters that can be represented using a single byte (the Latin letters A through Z, for example). Thus, the same information can be represented in graphic strings and mixed data strings. The only difference is in how the information is represented.

Graphic strings consist only of DBCS characters. There are no shift codes or SBCS characters. In mixed data strings, DBCS characters (if any) are delimited by shift codes and SBCS characters are used whenever possible. Since ASCII does not have shift-in and shift-out characters, this data type is not applicable to ASCII input.

If most or all of the characters must be represented with DBCS characters (because most of the characters are Kanji, for example), use the graphic data type. This saves you the two bytes of shift-in and shift-out characters it costs to switch from double-byte back to single byte. However, if most characters can be represented with SBCS characters, use mixed data to save one byte for every single-byte character in the column.

Numeric Data Types

Table 9 describes the data types for numbers.

Table 9. Numeric Data Types

Data Type	Denotes a Column of...
SMALLINT	Small integers. A small integer is an IBM System/390 two-byte binary integer of 16 bits; the range is -32768 to +32767
INTEGER or INT	Large integers. A large integer is an IBM System/390 fullword binary integer of 32 bits; the range is -2147483648 to +2147483647
REAL or FLOAT(*n*)	Single precision floating-point numbers. *n* must be in the range 1 through 21. If you omit *n*, the column has double precision. A single precision floating-point number is an IBM System/390 short floating-point number of 32 bits.
FLOAT, FLOAT(*n*), or DOUBLE PRECISION, or DOUBLE	Double precision floating-point numbers. *n* must be in the range 22 through 53. A double precision floating-point number is an IBM System/390 long floating-point number of 64 bits. The range of magnitudes for floating-point numbers of either type is about 5.4E-79 to 7.2E+75
DECIMAL(*p,s*) or DEC(*p,s*), or NUMERIC(*p,s*)	IBM System/390 packed decimal numbers with precision *p* and scale *s.* The precision *p,* which is the total number of digits including the digits following the decimal point, must be greater than 0 and less than 32. The scale *s*, which is the number of digits in the fractional part of the number, must be greater than or equal to 0 and less than or equal to the precision. *s* can be omitted; its default is 0. And if *s* is omitted, *p* can also be omitted; its default is 5. The maximum range is 1 - 10**31 to 10**31 - 1

For integer values, SMALLINT or INTEGER (depending on the range of the values) is preferable to DECIMAL or FLOAT. (But when using them in COBOL application programs, see the suggestions in Section 3 of *Application Programming and SQL Guide*; the range allowed by the COBOL PICTURE clause is not as large as the DB2 range.)

Date, Time and Timestamp Data Types

Table 10 describes the data types for dates, times, and timestamps.

Table 10. Date, Time and Timestamp Data Types

Data Type	Denotes a Column of...
DATE	Dates. A *date* is a three-part value representing a year, month, and day in the range 0001-01-01 to 9999-12-31
TIME	Times. A *time* is a three-part value representing a time of day in hours, minutes, and seconds, in the range 00.00.00 to 24.00.00
TIMESTAMP	Timestamps. A *timestamp* is a seven-part value representing a date and time by year, month, day, hour, minute, second, and microsecond, in the range 0001-01-01-00.00.00.000000 to 9999-12-31-24.00.00.000000

Advantages of Date/Time Data Types

Numbers representing dates and times can, of course, be stored in columns with numeric data types; if they include special characters as separators, they can be stored in string columns. But neither of those options allows the advantages described below, which belong to the data types DATE, TIME, and TIMESTAMP.

Variable Input and Output Format: Date/time values in DB2 are stored in a special internal format. When you load or retrieve data, DB2 can convert to or from any of the formats in the Table 11

Table 11. Date and Time Format Options

Format Name	Abbreviation	Typical Date	Typical Time
International Standards Organization	ISO	1986-12-25	13.30.05
IBM USA standard	USA	12/25/1986	1:30 PM
IBM European standard	EUR	25.12.1986	13.30.05
Japanese Industrial Standard Christian Era	JIS	1986-12-25	13:30:05

You also have the option of supplying an exit routine to make conversions to and from any local standard (choose LOCAL on the DATE FORMAT and TIME
| FORMAT fields on installation panel DSNTIP4). For instructions about writing and using a date or time exit routine, see “Date and Time Routines” on page X-51.

When loading date or time values from an outside source, DB2 accepts any of the
formats listed in Table 11, and converts valid input values to the internal format.
For retrieval, there is a default format that you select when installing DB2, but you
can override that default for every statement in an application program by a
precompiler option, or for particular instances by the scalar function CHAR. For
example, whatever your local default, the following statement displays employees'
birth dates in IBM USA standard form:

```
# SELECT EMPNO, CHAR(BIRTHDATE, USA) FROM DSN8510.EMP;
```

Queries Sent to a Distributed System: When an SQL statement is sent to a remote system, there are some restrictions for date and time formats. See Chapter 3 of *SQL Reference*for more information.

Comparing Data Types

DB2 compares values of different types and lengths provided that both values are numeric, both values are character strings, or both values are graphic strings. However, if two columns are compared (for example, to join tables) and their data types and lengths are not identical, DB2 might not use indexes that are defined on the columns. Therefore, when defining a column that will be compared with another column, ensure that their data types and lengths are identical.

Character strings are converted to the same coded character set prior to comparison.

You *cannot* make date and time comparisons with values of different types. You can compare a date only with a date, a time with a time, and a timestamp with a timestamp (or, in each case, with a valid string representation of a date, time, or timestamp).

If a column uses a field procedure, values to be compared to it are first encoded by the field procedure. If a column with a field procedure is compared to another column, that column must have the same field procedure. If the encoded values are numeric, their data types must be identical; if they are strings, their data types must be compatible.

Chapter 2-5. Designing Indexes

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

DB2 uses indexes not only to enforce uniqueness on column values, as for parent keys, but also to cluster data, partition tables, and to provide access paths to data for queries. Understanding some of the structure of DB2 indexes can be important for achieving your best performance.

Even though a table has indexes, DB2 does not always use one to access the data. You can tell whether an index is used in executing a particular SQL statement by using the EXPLAIN statement, or the EXPLAIN option of bind. For a description of this technique, see “Chapter 5-10. Using EXPLAIN to Improve SQL Performance” on page 5-261.

Index Types and Recommendations

There are two types of indexes in DB2, type 1 and type 2. Use the statement CREATE INDEX to create either type. To change an index from one type to the other, use ALTER INDEX; then run the utility RECOVER INDEX to complete the change.

When you use CREATE INDEX, the default value of the index type depends on the value of LOCKSIZE for the associated table space:

1. If LOCKSIZE is ROW then the default index type is type 2. You cannot use row locking with a type 1 index.
2. If LOCKSIZE is not ROW, then the default for CREATE INDEX is the type specified in field DEFAULT INDEX TYPE of installation panel DSNTIPE.

| The storage space required for an index depends on several factors. For more
| information on calculating the estimated space required for a Type 2 index, see
| *Installation Guide*.

To convert an existing index to type 2, use the ALTER INDEX statement. After using ALTER INDEX, run the utility RECOVER INDEX to complete the task.

Type 2 Indexes

Type 2 indexes differ from type 1 indexes in that:

- Type 2 indexes have no subpages.
- Type 2 indexes require no locks on their pages. A lock on the data page or row locks the index key.

  Type 2 indexes support high concurrency by locking the underlying data page or record. Only one lock is held after an update to a data row or page and its associated indexes.

  (In some cases, this might produce contention that does not occur with a type 1 index. For example, a type 2 index scan might require locks on data pages, but a type 1 index scan does not. Hence, the index scan might contend with an update operation.)

- With type 2 indexes, no locks are acquired on index pages during modifications of index structures. A *structure modification* (such as page splits or page deletions) is an index operation that updates the nonleaf portion of the index tree. and changes the structure of the tree. Other transactions can read or modify pages involved in structure modifications.
- Type 2 indexes use suffix truncation on nonleaf pages to store the high key values. Only enough of the prefix portion of the key is stored to distinguish it from the previous key. For example, suppose that during a leaf page split, the first key value on the new page is WXYZ4792 and the last key value on the previous page is WXYY5341. Without suffix truncation, WXYZ4792 would be stored as the new high key value in the nonleaf page. With suffix truncation, only WXYZ is stored as the new high key value. Depending on the nature of the index key (if most of the keys are alike) there could be a reduction in the number of index levels and, consequently, in the number of pages read and in the cost of I/O for index access.
- The record identifiers (RIDs) stored with a type 2 nonunique index entry are kept in ascending order.

  This RID ordering is maintained so that a binary search can locate the RID of interest efficiently. RID maps for type 2 indexes can contain as many RIDs as will physically fit on the page; they are not limited to 256 entries, as in type 1. Deleting a single duplicate no longer requires a slow sequential search of duplicates that can span multiple leaf pages. RID ordering improves performance in deleting or updating entries, when the number of duplicates is in the thousands.
- Unless the table or table space is locked in mode X, a DELETE operation does not physically remove the entry. Instead, it pseudo-deletes the entry by leaving it in the page, but marks it for deletion. The space remains available in the index at least until the data is committed, so if a rollback takes place, there will always be enough room to reinsert the deleted entry and avoid the need to split the page. After the deleting operation commits, the committed pseudo-entry can be physically deleted when space is needed for an insert, or when the percentage of pseudo-deleted entries marked on a page reaches a certain threshold.
- There are four types of index pages: the header page, space map page, leaf page and nonleaf page. With type 2 indexes, the formats of these pages change. For example, type 2 space maps cover fewer pages, but contain more information per page, than type 1 space maps.
- There are no one-level type 2 indexes. A type 2 index is created as a two-level index with a root page that points to an empty leaf page.
- With a type 2 index, the predicate can be evaluated when the index is accessed, particularly if all columns in the predicate are in the index. If not all columns are present in the index, some evaluation might still be possible when the index is accessed, and then the predicate is evaluated further when the data is accessed.

Use type 2 indexes whenever possible. Type 2 index provide increased concurrency and performance. They also provide the following functions, which are unavailable with type 1 indexes.

- Row locking on a table space
- UR isolation for an access path

- Processing of queries by multiple parallel tasks
- Concurrent access to separate logical partitiions

Leaf Pages, Root Page, and Subpages

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, containing entries that point to the leaf pages. If it has more than one nonleaf page, the nonleaf pages whose entries point directly to leaf pages are said to be on *level 1*; there must be a second level of nonleaf pages to point to level 1, and so on. The highest level contains a single page, called the *root page*. When DB2 first builds your index, it creates an index tree which resides in a 4KB index page, called the *root page*. This index tree points directly to the data in your tables giving the key and the row ID. A typical index is shown schematically in Figure 35.

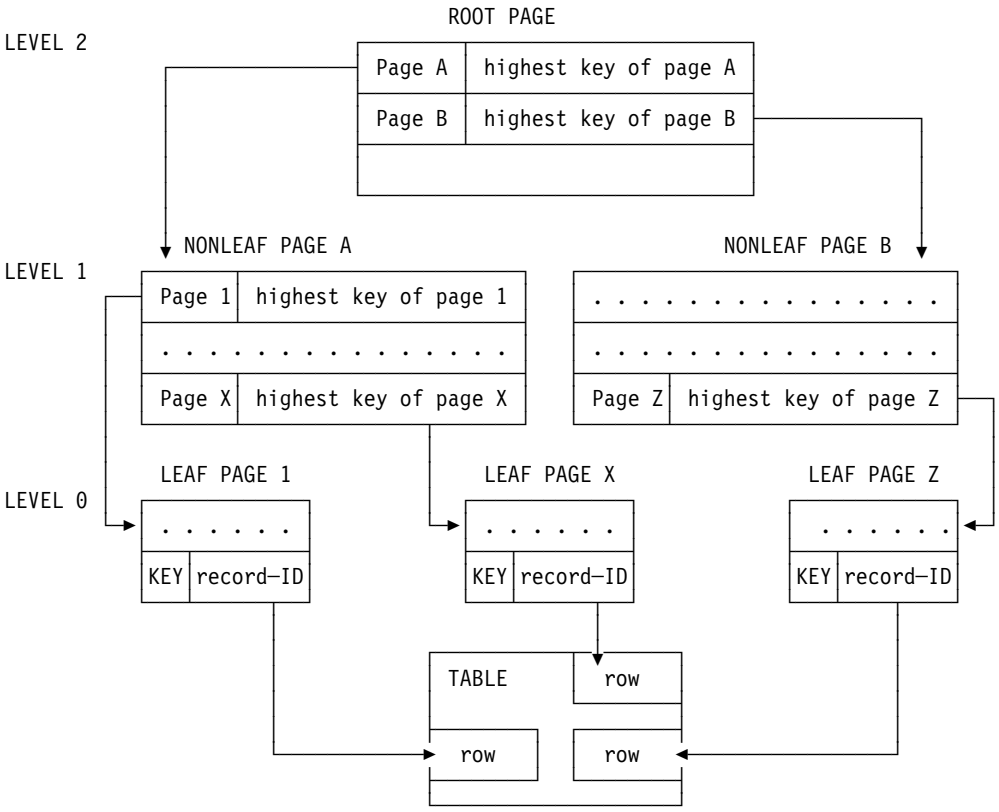


Figure 35. Sample Index Structure and Pointers (Three-Level Index)

Type 1 Indexes and Locking

The leaf pages of a type 1 index can be subdivided into 1, 2, 4, 8, or 16 *subpages*; the default is 4.

The subpage is a unit of locking. The purpose of having more than one subpage is to increase concurrency for applications that update, rather than insert. Insert applications often must reduce the number of subpages for concurrency. Although increasing the number of subpages is effective in reducing lock contention, the

processing overhead is higher, and there is also a fixed storage overhead for each subpage.

If your type 1 index grows beyond the 4K index page, a page split occurs and a level 1 index page is created. After the first page split, the root page is no longer called a leaf page; the level 0 pages pointing directly to the data are now known as leaf pages. Each leaf page holds approximately 4096 bytes of data, and, if one page is full, an insertion in the index causes a leaf page split. A page split adds another page to the same level of the index. If there is no room on that level of the index tree to accommodate another page, another level of index is added.

The number of pages in a level of the index depends on the number of index entries you have and the length of the key. The root page contains the page number and highest key of each page in the next level of the index. Similarly, the nonleaf pages at a given level contain the page number and highest key of each page in the next level. For an explanation of what a key is, refer to "Index Keys." Figure 35 on page 2-53 shows an index that has split into three levels, where the index entries in the root page point to the entries in the level 1 index page.

For a nonpartitioned index, there is one tree for each index. For a partitioned index, there is one tree for each partition.

Type 2 Indexes and Locking

A possible solution to the concurrency problem is to use type 2 indexes, which allow locking every row in the table separately (LOCKSIZE ROW). Type 2 indexes do not have subpages. They do not lock index pages. See "Type 2 Indexes" on page 2-51 for a description of the differences between type 1 and type 2 indexes.

Index Keys

A table can have more than one index, and an index key can use one or more columns. An *index key* is a column or an ordered collection of columns on which an index is defined. A *composite key* is a key built on 2 to 64 columns.

The usefulness of an index depends on its key. Columns that you use frequently in performing selection, join, grouping, and ordering operations are good candidates for use as keys.

The number of bytes in an index key is the sum of the number of bytes in the column or columns that make up the key, plus one byte for each column that can contain null values. (For columns with a field procedure, use the number of bytes in the encoded field, not the number in the decoded column. This distinction is explained under "Field Procedures" on page X-57.) For information about restrictions on key length, see Chapter 6 of *SQL Reference*.

Some Things to Remember When Defining Keys:

- **Column updates require index updates**.

  Define as few indexes as possible on a column that is updated frequently, because every change must be reflected in each index.

- **A composite key might be more useful than a key on a single column when the comparison is for equality**.

A single multicolumn index is more efficient when the comparison is for equality and the initial columns are provided. For example, if an index is composed of columns A, B, and C, a SELECT statement with a WHERE clause of the form WHERE A = *value* AND B = *value* might be processed more efficiently than if there are separate indexes on A and on B.

However, for more general comparisons, such as A > *value* AND B > *value*, multiple indexes might be more efficient.

- **Indexes are important tools for improving performance**.

  To fully understand how indexes are used during access path selection, see "Chapter 5-10. Using EXPLAIN to Improve SQL Performance" on page 5-261.

Using Unique Indexes

DB2 allows you to enter duplicate values in a key column. If you do not want duplicate values, use CREATE UNIQUE INDEX. For example, in the sample application it is important that there be no duplicate department IDs in the department table. Creating a unique index, as in the following example, prevents duplicates.

```
CREATE UNIQUE INDEX XDEPT1
  ON DSN8510.DEPT (DEPTNO);
```

The index name is XDEPT1 and the indexed column is DEPTNO.

If a table has a primary key (as the department table has), its entries must be unique. This uniqueness is enforced by defining a unique index on the primary key columns, with the index columns in the same order as the primary key columns.

But it is not necessary that all keys be unique. For example, an index on the SALARY column of the employee table would allow duplicates, because it is perfectly reasonable for several employees to earn the same salary.

Before you create a unique index on a table that already contains data, be sure that no pair of rows has the same key value. If DB2 finds a duplicate value in a set of key columns for a unique index, it returns an error message and does not create the index.

Using the UNIQUE WHERE NOT NULL Clause

A single-column key defined with UNIQUE WHERE NOT NULL can have more than one null value, but its nonnull values must be unique. For example, suppose some of your employees have paging phones, and their phone numbers are recorded in column BEEPER of the table of employee data. An index on BEEPER, defined with UNIQUE WHERE NOT NULL, allows many null values for employees with no paging phone, but ensures that any number added or updated is unique.

For a key on more than one column, where the index is defined with the UNIQUE WHERE NOT NULL clause, the key value is null if any part of the key value is null. An index defined with UNIQUE WHERE NOT NULL must be type 2.

Using Composite Keys

You can create an index with a composite key, containing multiple columns. The index entries can be in sequence by the ascending values or descending values of the columns. The keywords ASC and DESC of the CREATE INDEX statement indicate ascending and descending order. ASC is the default and can be omitted. The other clauses of the CREATE INDEX statement are described on following pages.

```
CREATE UNIQUE INDEX DSN8510.XPROJAC1
  ON DSN8510.PROJACT
     (PROJNO ASC,
     ACTNO ASC,
     ACSTDATE DESC)
     ...
```

Clustering Indexes

To specify a clustering index, use the CLUSTER clause in the CREATE INDEX statement. When a table has a clustering index, an INSERT statement inserts records as nearly as possible in the order of their index values. These clustered inserts can provide a significant performance advantage in some operations, particularly those that involve many records, such as grouping, ordering, and comparisons other than equal. Although a table can have several indexes, only one of them can be a clustering index.

If you first create a nonclustering index, DB2 organizes the data by that index. If later you create an index you specify as clustering, DB2 identifies it as the clustering index but does not rearrange the data already in the table. The data is still organized by the original nonclustering index you created for the table. However, when the table space is reorganized by the REORG utility, DB2 clusters the data in accordance with the new clustering index. Therefore, if you know you want a clustering index, it is better to define it before you load the table. If that is not possible, define the index and then reorganize the table.

Partitioned Indexes

When you store a table in a partitioned table space, you tell how to divide the data among partitions by using the PART clause of a CREATE INDEX statement. Therefore, the index that divides the data is a partitioned index. It is also a clustering index, because the data is clustered by the index key values. Thus, your PART clause must be preceded by CLUSTER. The index must be type 2 to support logical partitions.

Although the total length of your partitioned index columns can be up to 255 bytes (less one for each column that can contain nulls), only the first 40 bytes are actually used to partition the data. DB2 is not sensitive to field boundaries when it determines partitioning. There might be situations in which you need to know where the 40-byte truncation occurs. For example, you might not want to truncate a field with zeros in the high order bytes, if that could disrupt your partitioning strategy.

Recommendation: choose values for partitioning keys that are unlikely to be updated. Updates that cause a row to move from one partition to another have lower performance. To prevent degrading the performance of other SQL statements that are in the same unit of work, update partitioning keys in a separate unit of

work, if possible. If you must move many rows into another partition, use the
procedure in 2-150.

Nonpartitioned Indexes

You can define a nonpartitioned index on either a nonpartitioned or a partitioned table space. If you define a nonpartitioned index and a partitioned index on the same table, you lose some of the benefits of partition-level independence for utility operations because access to a nonpartitioned index must be sequential. However, logical partitions of a nonpartitioned index can usually be accessed by utilities and SQL applications concurrently. To take advantage of this concurrency, your nonpartitioned indexes must be type 2. For more information about utility operations, nonpartitioned indexes, impacts of using type 1 or type 2 indexes, see “Utility Operations with Nonpartitioned Indexes” on page 5-190.

Designing Index Spaces

CREATE INDEX creates an index space in the DB2 database containing the table on which the index is defined. If you do not defer the index build, a populated table space is always scanned during the execution of CREATE INDEX. Because the LOAD utility builds indexes more efficiently, it is best to create the indexes before loading the table.

If you are creating a nonunique index on a populated table, consider using the DEFER clause, as described in “Creating an Index on a Large Table” on page 2-104. This allows you to build the index later using RECOVER INDEX.

When you use CREATE INDEX, specify the USING clause. If you use a storage group, consider specifying the primary and the secondary space allocation quantities (PRIQTY and SECQTY). You specify the allocation as a number of kilobytes; DB2 specifies the value to VSAM as a number of pages. For each clause, the default value is three pages. For more information about how space allocation can affect the performance of mass inserts, see “Ensure Allocation in Cylinders” on page 5-40.

The primary quantity is particularly important; so try to specify a value that avoids secondary allocation without wasting space. DB2 can extend a nonpartitioned index space to more than one data set, but only if the value of (PRIQTY + 118 × SECQTY) is at least the value of the PIECESIZE parameter. (You might possibly reach 123 extents before reaching the limit, but 119 is more likely.) To make the best use of space for nonpartitioned indexes, choose a value for PIECESIZE that is an even multiple of (PRIQTY + 118 × SECQTY).

The default value of SECQTY is 10 percent of the specified PRIQTY, or three times the page size, whichever is larger.

If you partition the table space, the clustering index is also partitioned.

You can use the USING clause to specify space for the entire index, or, if the index is partitioned, you can specify 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.

Chapter 2-6. Designing Table Spaces

After you design a relational database, you must create DB2 objects. This chapter includes physical design considerations and implementation information.

Deciding What Type of Table Space and How Many

The information under this heading, up to “Building the Compression Dictionary” on page 2-64, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

You can choose from three types of table spaces:

Simple A table space that can contain more than one table. The space is composed of pages, and each page can contain rows from many tables.

Segmented A table space that can contain more than one table. The space is composed of groups of pages called segments. Each segment is dedicated to holding rows of a single table.

Partitioned A table space that can contain only a single table. The space is subdivided into partitions based on the key range of a nominated partitioning index. Each such partition can be processed concurrently by utilities and SQL.

If you are creating a single table, you can choose any of the three types of table spaces. If you are creating several tables in a table space, you can choose either a simple or segmented table space.

Rule of Thumb: All large table spaces should be partitioned and the rest of the table spaces should be segmented. Use the following description of the three types of table spaces to help you decide.

Simple Table Spaces

In general, use segmented or partitioned table spaces for most of your DB2 data. The following section describes why you might use simple table spaces.

Advantage of Simple Table Spaces

If you know how your data is usually retrieved and you don't insert or delete rows in your table too often, you can put things together physically that belong together logically. For example, for logically hierarchical data in which dependents are often retrieved with their parent, you might want to load the data in hierarchic order to improve the chances of parent and dependent rows residing on the same page.

The *disadvantages* of simple table spaces are as follows:

- It takes more time to scan for all the rows of a table when tables are intermixed in a table space. Scanning one table requires scanning the entire table space.
- When a table is dropped, the space occupied by the table does not become available for reuse immediately.
- The entire table space is unavailable when certain DB2 utilities are executing.

- For simple table spaces containing more than one table, there is no provision for locking a single table. If LOCK TABLE is specified for a table that is contained in a simple table space, all tables in the table space are actually locked.
- A page lock in a simple table space can mean that information from other tables is locked as well. For further information on locking, refer to “Chapter 5-7. Improving Concurrency” on page 5-137.

Segmented Table Spaces

A segmented table space is ideal for storing more than one table, especially relatively small tables. The pages of the space are organized into *segments*, and each segment holds records from only one table. Each segment contains the same number of pages, which must be a multiple of 4 (from 4 to 64). Each table uses only as many segments as it needs, and a space map records the assignments of segments to tables.

As records are inserted by INSERT or LOAD, there is no attempt to keep segments from the same table together. Figure 36 shows one use of space that might result. Reorganizing the table space puts segments of the same table together.

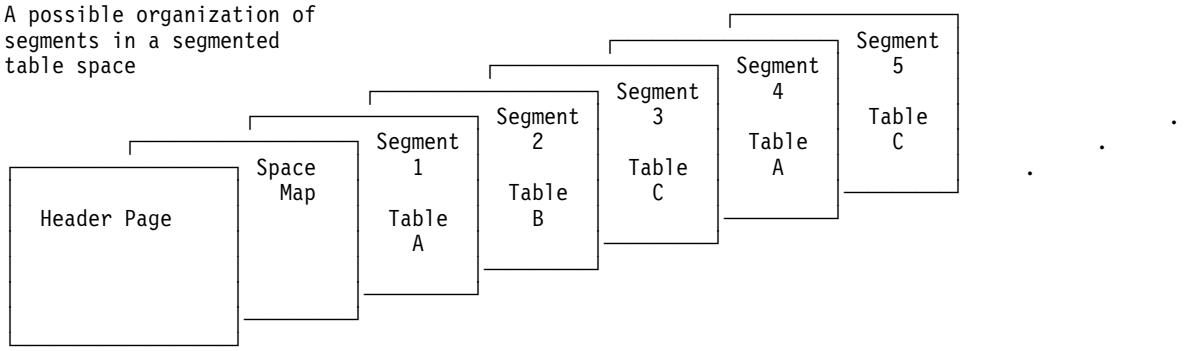


Figure 36. Segments of a Segmented Table Space

Advantages of Segmented Table Spaces

- When scanning a table, only the segments assigned to that table need be accessed. Pages of empty segments do not have to be fetched.
- When locking a table, the lock does not interfere with access to segments of other tables.
- When a table is dropped its segments become available for reuse immediately after the DROP is committed; it is not necessary to wait for an intervening REORG utility job.
- When all rows of a table are deleted (mass delete), all segments but the first become available for reuse immediately after the delete has been committed; it is not necessary to wait for an intervening REORG utility job.
- A mass delete of all rows of a table operates much more quickly and produces much less log information.

  However, if you have a validation procedure, a parent table in a referential constraint, or DATA CAPTURE CHANGES defined on a table in a segmented table space, each row has to be processed; thus, you lose the advantage of the quick mass delete.

- Even if the table space contains only one table, segmenting it means that COPY does not copy pages that are empty because of a dropped table or a mass delete.
- Because the space map contains more information than a space map for a simple table space, some read operations are avoided when inserting records.

Creating fewer table spaces, by storing several tables in one table space, can help you avoid reaching the maximum number of concurrently open data sets. Each table space requires at least one data set, and there is a maximum number of concurrently open data sets that is determined during installation. Using fewer table spaces means less time spent in data set allocation and deallocation.

Disadvantages of Segmented Table Spaces

- Some DB2 utilities, such as LOAD with the REPLACE option, RECOVER, COPY, and REORG, operate only on a table space or a partition. Therefore, for a segmented table space, you must run these utilities on the entire table space. For a large table space, this can have a negative impact on availability.
- There is some additional overhead to maintaining the space map.

If you choose to segment the table space, use the SEGSIZE clause. Refer to "Creating a Segmented Table Space" on page 2-90 for more information.

Partitioned Table Spaces

You use a partitioned table space to store a single table. As shown in Figure 37, each partition contains one part of the table. We recommend that if you implement a partitioned table space, create it as LARGE to use its advantages.

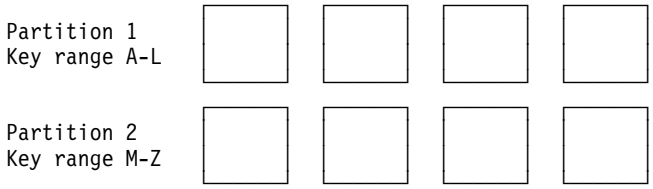


Figure 37. Pages in a Partitioned Table Space. The first page of each partition is a header page. The second page is a space map page.

A partitioned table space needs a clustering index because rows are assigned to the partitions according to the values of their cluster index keys.

Advantages of Partitioned Table Spaces

Partitioned table spaces have the following advantages:

- You can spread a large table over several DB2 storage groups or data sets. All the partitions of the table do not have to use the same storage group.
- You can work on part of your data at a time, and possibly decrease the amount of time required for the operation. Partitioned table spaces let a utility job work on part of the data at a time, while allowing concurrent access to other jobs or applications on other partitions. In that way, several concurrent utility jobs can, for example, load all partitions of a table space concurrently.
- You can break mass update, delete, or insert operations into separate jobs, each of which works on a different partition. Breaking the job into several smaller jobs that run concurrently can reduce the time for the whole task. If

your table space uses nonpartitioned indexes, you might need to modify the size of the data sets in the indexes to avoid I/O contention among the concurrently running jobs. Use the PIECESIZE parameter of CREATE or ALTER INDEX to modify the sizes of the index data sets.

- You can put frequently accessed data on faster devices. If a clustering (partitioned) index can separate more frequently accessed data from the remainder of the table, that data can be put in a partition of its own and can use a different device type.
- You can take advantage of parallelism for certain read-only queries. When DB2 determines that processing will be extensive, it can begin parallel processing of more than one partition at a time. Parallel processing (for read-only queries) is most efficient when you spread the partitions over different DASD volumes and allow each I/O stream to operate on a separate channel. For more information about parallelism, see "Parallel Operations and Query Performance" on page 5-299.

  You can take advantage of query parallelism by using the Parallel Sysplex data sharing technology to process a single read-only query across many DB2s in a data sharing group. Sysplex query parallel processing is optimized when each DB2 is on a separate central processor complex. For more information about Sysplex query parallelism, see *Data Sharing: Planning and Administration*.

Advantages of Large Table Spaces

Large table spaces offer flexibility in database design:

- You do not have to separate large amounts of data into different tables because of size constraints.
- Greater amounts of data are available when utilities are run because you distribute data across more partitions. Larger numbers of partitions can also speed up parallel queries by permitting more concurrent parallel processing.
- You can better accommodate a growing database.

Disadvantages of Partitioned Table Spaces

- You must have a partitioning key and a clustering index on a fixed number of partitions. This means you must define all the partitions you expect to use even if you will not use them for some time.
- Table space scans can be less efficient than table space scans of segmented table spaces. See "Table Space Scans (ACCESSTYPE=R PREFETCH=S)" on page 5-275 for more information.
- DB2 opens more data sets when you access data in a partitioned table space than when you access data in other types of table spaces.

See "Creating a Partitioned Table Space" on page 2-91 for more information on partitioned table spaces.

Use LOCKSIZE with Performance in Mind

LOCKSIZE ANY is the default for CREATE TABLESPACE, allowing DB2 to choose the locksize, usually LOCKSIZE PAGE and LOCKMAX SYSTEM. Before you choose LOCKSIZE TABLESPACE or LOCKSIZE TABLE, you should know why you do not need concurrent access to the object. Before you choose LOCKSIZE ROW,

estimate the increase in overhead for locking and weigh it against the increase in concurrency.

For more information see “Recommendations for Database Design” on page 5-141.

Compressing Data in a Table Space or Partition

Specifying COMPRESS YES on CREATE or ALTER TABLESPACE when used with a LOAD or REORG utility compresses data in a table space or partition. 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.

Deciding Whether to Compress

Consider the following before compressing a table space or partition:

- Saving space in large table spaces

  Compression can work very well for large table spaces. With small table spaces, the size of the compression dictionary (8KB to 64KB) can offset the space savings provided by the compression.

- The size of the data rows

  DB2 compresses the data of one record at a time. (The prefix of the record is not compressed.) However, if compressing the record produces a result that is no shorter than the original, DB2 does not compress the record. This is one reason why you should *not* try to use data compression to encrypt data.

  The closer that the average row length is to the actual page size, the more inefficient compression might be; because rows cannot span pages, you might not achieve enough compression to fit two rows on a page.

- Processing costs

  Compressing data costs processing time. The cost is significantly lower if you use IBM's synchronous data compression hardware than if you use the DB2-provided software simulation. DB2 determines at startup time whether you have hardware support for compression. If you do not, DB2 uses software simulation.

  It costs less to decompress data than to compress it, and the overall cost depends on the patterns in your data. You can use the DSN1COMP stand-alone utility to find out how effective compressing your data will be. For more information, see Section 3 of *Utility Guide and Reference*.

  See “Compressing Your Data” on page 5-102 for more performance considerations.

- 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

  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 the compressibility of your data.

  If you have a table space with more than one table, and if 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, we recommend that you put a table you want to compress into a table space by itself, or into a table space containing only tables with similar kinds of data.

  REORG, on the other hand, 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 might produce better results for table spaces containing tables with dissimilar kinds of data.

- 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, we recommend that you do not use DB2 data compression in conjunction with DSN8HUFF. This adds little additional compression at the cost of significant extra CPU processing.

- Effects on logging

  If a data row is compressed, all data 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. In any event, applications that are sensitive to log-related resources can see 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. Using IFCID 306, however, you can cause DB2 to write log records of compressed data in decompressed format. You can retrieve those by using the IFI function READS.

- Distributed data

  DB2 decompresses data before transmitting it to VTAM.

Building the Compression Dictionary

Each table space or partition that contains compressed data has a *compression dictionary*. The compression dictionary contains information used to control compression and decompression. The dictionary contains a fixed number of entries, usually 4096, and resides with the data. Its 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.

You can build the compression dictionary by using the LOAD utility with the REPLACE or RESUME NO options, or by using the REORG TABLESPACE utility. With LOAD, the dictionary is built as data is loaded; no data is compressed until the dictionary is completely built. For more information about using LOAD or REORG to create a compression dictionary, see Section 2 of *Utility Guide and Reference*.

Determining the Effectiveness of Compression

There are a couple ways for you to determine how effective data compression is by using compression reports and catalog statistics. If you want to estimate the compression effectiveness prior to compressing the data, then use the DSN1COMP stand-alone utility.

Compression Reports

You can determine the effectiveness of compression 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 gives you information about the amount of compression and how much space is saved. (REORG with the KEEPDICTIONARY option does not produce the report.)

Catalog Statistics

In addition to these reports, the PAGESAVE column of the SYSIBM.SYSTABLEPART catalog table and the PCTROWCOMP columns of SYSIBM.SYSTABLES and SYSIBM.SYSTABSTATS contain information about data compression. PAGESAVE tells you a percentage of pages saved by compressing the data, and PCTROWCOMP 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.

DSN1COMP

Use the 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 containing 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 Section 3 of *Utility Guide and Reference* .

Chapter 2-7. Designing Storage Groups and Managing 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/MVS's storage management subsystem (DFSMS). A DB2 storage group is a named set of DASD volumes, in which DB2 does the following:

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

This chapter explains how to manage your DB2 data sets.

Naming DB2 Storage Groups and Databases: 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	DSN8G510
Database	DSN8D51A

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

If you are using shared read-only data, see “Appendix F. Sharing Read-Only Data” on page X-153.

Managing Your DB2 Data Sets with DFSMShsm

If you decide to use DFSMShsm for your DB2 data sets, you should develop a migration plan with your system administrator. With user-defined 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.

Do not let DFSMShsm migrate the DB2 catalog, DB2 directory, active logs, and the temporary database (DSNDB07 in a non data-sharing environment) before starting DB2. Considerations for using DFSMShsm for archive log data sets are discussed in “Archive Log Data Sets” on page 4-91.

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.

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 does not extend a data set the same way DB2 does, so you must alter the page set to contain extends defined by DB2. Use ALTER TABLESPACE to enlarge the primary and secondary values for stogroup-defined data sets, because DB2 could run out of extends when REORG or LOAD REPLACE (unloading and reloading the same data) is used.

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

For user-defined data sets (see “Managing Your Own DB2 Data Sets”), define the data sets with larger primary and secondary values.

For more information about using DFSMShsm to manage DB2 data sets, see *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide.*

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 the individual data sets on the volumes. (Although you can do 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

Managing DB2 auxiliary storage yourself involves using access method services directly. 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 nonpartitioned index
- A partition of a partitioned 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 DASD volumes that are attached on separate channel paths. For more information about data set extension, see “Extending DB2-Managed Data Sets” on page 5-100.

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 meet these requirements:

1. Define the data sets *before* you issue the CREATE TABLESPACE or CREATE INDEX statement.
2. Give each data set a name with this format:

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

   The name variables are defined below:

   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 will 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*.)

   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 defined 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 space runs short, DB2 issues a warning message. If the size of the data set for a simple or segmented table space approaches 2 gigabytes, define another data set. Give it the same name as the first data set, and the number 002. The next data set will be 003, and so forth. You might eventually need up to 32 data sets (the maximum) for a simple or segmented table space.

For table spaces, it is possible to reach the 119-extent limit for the data set before reaching the 2-gigabyte limit for a nonpartitioned table space or the 4-gigabyte limit for a partitioned table space. If this happens, DB2 does not extend the data set.

3. You must 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. If you specify passwords when defining a VSAM data set, give your highest-level password in the DSETPASS clause (in the CREATE TABLESPACE or CREATE INDEX statement).
5. Define the data sets as LINEAR. Do not use RECORDSIZE or CONTROLINTERVALSIZE; these attributes are invalid, and are replaced by the specification LINEAR.
6. Use the REUSE option. You must define the data set as REUSE in order to use the DSN1COPY utility.
7. Use SHAREOPTIONS(3,3).

The DEFINE CLUSTER command has many optional parameters that do not apply when the data set is used by DB2. 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 OWNER parameter value for clusters 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 yourself 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, destroying the old data.

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

Defining Data Sets for Online REORG

Before executing the REORG utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE, you must create shadow data sets, with names that have the form `catname.DSNDBx.dbname.psname.S0001.Annn`.

In the switch phase of REORG with SHRLEVEL REFERENCE or SHRLEVEL CHANGE, REORG exchanges the names of the original data sets and shadow data sets by renaming the data sets as follows:

1. Rename the original data set from `catname.DSNDBx.dbname.psname.I0001.Annn` to `catname.DSNDBx.dbname.psname.T0001.Annn`, which is a temporary name.

2. Rename the shadow data set from `catname.DSNDBx.dbname.psname.S0001.Annn` to `catname.DSNDBx.dbname.psname.I0001.Annn`.
3. Rename the original data set from `catname.DSNDBx.dbname.psname.T0001.Annn` to `catname.DSNDBx.dbname.psname.S0001.Annn`.

Therefore, data sets with names that have the form `catname.DSNDBx.dbname.psname.T0001.Annn` must not already exist when you invoke the REORG utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE.

In addition, before executing the REORG utility with SHRLEVEL REFERENCE or SHRLEVEL CHANGE on partition *mmm* of a partitioned table space, you must create (for each nonpartitioned index) a shadow data set for a copy of the logical partition of the index, with a name that has the form `catname.DSNDBx.dbname.psname.S0mmm.Annn`.

DEFINE CLUSTER Command

Figure 38 shows the DEFINE CLUSTER command, which defines the VSAM linear data set (LDS) for the DB2 catalogs SYSUSER table space in database DSNDB06. Assume that an integrated catalog named DSNCAT has already been defined.

```
DEFINE CLUSTER-
       (NAME(DSNCAT.DSNDBC.DSNDB06.SYSUSER.I0001.A001) -
        LINEAR                                         -
        REUSE                                          -
        VOLUMES(DSNV01)                                -
        MASTERPW(DBADMIN)                              -
        CONTROLPW(DBADMIN)                             -
        RECORDS(10 10)                                 -
        SHAREOPTIONS(3 3) )                            -
      DATA                                             -
       (NAME(DSNCAT.DSNDBD.DSNDB06.SYSUSER.I0001.A001) -
        MASTERPW(DBADMIN)                              -
        CONTROLPW(DBADMIN) )                           -
    CATALOG(DSNCAT/DSNDEFPW)
```

Figure 38. Defining a VSAM linear data set with DEFINE CLUSTER.

```
DEFINE CLUSTER-
       (NAME(DSNCAT.DSNDBC.DSNDB06.SYSUSER.S0001.A001) -
        LINEAR                                         -
        REUSE                                          -
        VOLUMES(DSNV01)                                -
        MASTERPW(DBADMIN)                              -
        CONTROLPW(DBADMIN)                             -
        RECORDS(24 12)                                 -
        SHAREOPTIONS(3 3) )                            -
      DATA                                             -
       (NAME(DSNCAT.DSNDBD.DSNDB06.SYSUSER.S0001.A001) -
        MASTERPW(DBADMIN)                              -
        CONTROLPW(DBADMIN) )                           -
    CATALOG(DSNCAT/DSNDEFPW)
```

Figure 39. Defining a shadow data set with DEFINE CLUSTER.

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

Chapter 2-8. Designing a Database in a Distributed Environment

Many businesses have a need to manage data from a variety of sources and locations. A distributed data processing environment allows more flexibility in the allocation of resources.

This chapter describes the planning considerations for accessing data in a distributed environment, including the implications for application and systems programming, and discusses stored procedures and the role of the DB2 administrator.

Ways to Access Distributed Data

From an application's point of view, there are two ways to access distributed data: one is called *DRDA access*, and the other is called *DB2 private protocol access*. Both of these methods are described in this section.

DRDA Access

DRDA access allows access to any remote database management system that implements the Distributed Relational Database Architecture (DRDA). An *application requester* can select from or update tables at a remote location (the *server*) by connecting to that location before executing SQL statements. We call this DRDA access because an application process can issue explicit or implicit SQL CONNECT statements; there is no use of three-part table or view names.

A package that you execute at another server can contain most kinds of SQL statements, for example, CREATE, SELECT, or even GRANT and REVOKE. It cannot contain statements to connect to yet another server. However, packages can contain statements with three-part names that allow them to access data at another DB2 using DB2 private protocol access. (This is sometimes known as a “double hop.”)

Using DRDA access, you can update at more than one system if you use a type 2 CONNECT statement. You can update at only one system if you use a type 1 CONNECT statement, or if the system is at a level of DRDA that does not support two-phase commit. Information about the CONNECT statement is in Section 6 of *Application Programming and SQL Guide* .

Additionally, applications using DRDA can use DB2's stored procedures. More information about stored procedures in a distributed environment is in “Stored Procedures” on page 2-76.

DB2 Private Protocol Access

DB2 private protocol access allows access only to other DB2 subsystems for MVS. An application can issue SQL SELECT, INSERT, UPDATE, or DELETE statements for a table at a remote location by using a three-part name, or an alias of that name, in which one part names the location at which the table resides. You cannot create or drop a table at a remote location using DB2 private protocol access.

This method is called DB2 private protocol access because DB2 determines where to connect to based on the name of the object in the statement. If you are a database administrator, you will probably be involved in maintaining the association of object names with locations.

Using DB2 private protocol access, you can update at multiple DB2 subsystems in the same unit of work. However, if you update data in a system using DB2 Version 2, you can update only that one subsystem in a single unit of work, although you can read from many other subsystems.

Figure 40 shows a comparison of the two ways to access distributed data. Which method you use depends on your particular application.

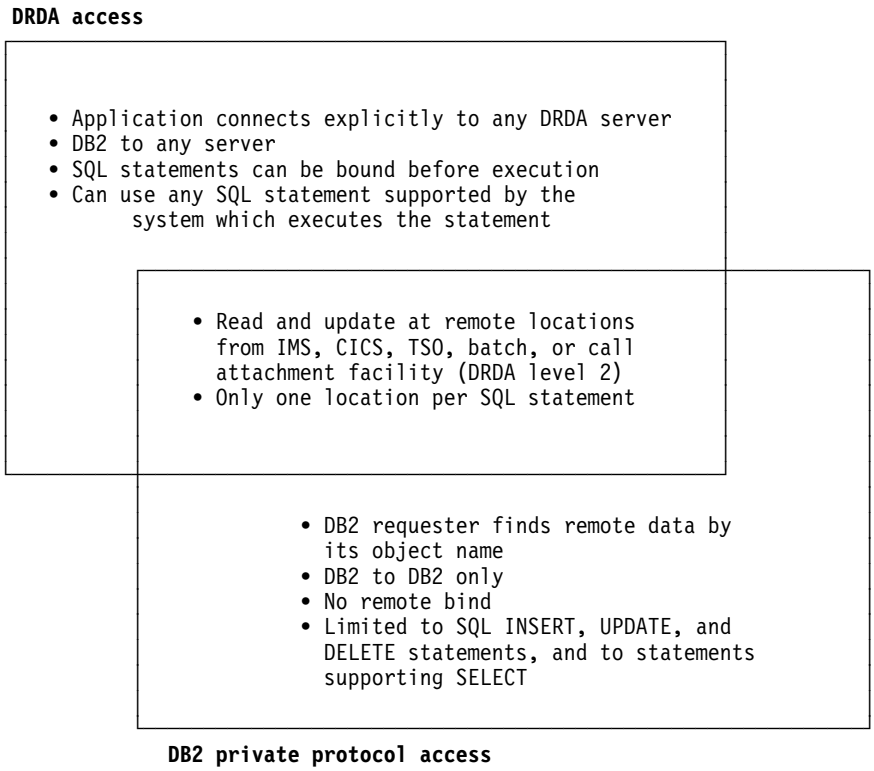


Figure 40. A Comparison of Ways to Access Distributed Data. The ability to update at remote locations through CICS and IMS requires communications and DRDA support for two-phase commit.

It is possible to use both types of distributed access in a single application. For information about planning a DB2 application to use distributed data, see Section 5 of *Application Programming and SQL Guide*. For more information about distributed data, see *Distributed Relational Database Architecture: Evaluation and Planning Guide*.

Coordinated Updates

Both DRDA access and DB2 private protocol access can coordinate updates across locations. Assuming both requesters and servers have communications and DRDA support for two-phase commit, you can both read from and update many remote locations in a single unit of work, no matter where your application originates. When your application reaches a commit point, it issues a single statement to commit or roll back the changes everywhere.

See Section 3 of *Installation Guide* for information on how to configure DB2's communication support for two-phase commit.

Implications for Application Programming

Implications of distributed data for application programming fall under four main headings. For detailed information about distributed processing, see *Application Programming and SQL Guide*.

- Naming conventions

  Names of tables and views can be qualified by a location name. When the current server is a DB2 subsystem, you can use the location name to access an object in another DB2 subsystem through DB2 private protocol access. You can also create aliases for tables or views in other systems. “Naming Remote Objects for DB2 Private Protocol Access” on page 2-80 and Section 2 of *Application Programming and SQL Guide* address the topic of naming conventions.

- Access limitations

  Refer to Figure 40 on page 2-74 for a general look at what each method of access can do. The limitations on access are based also on the level of DRDA that the other server or requester supports. We assume here that both server and requester are using DRDA and both support the ability to update at multiple sites (two-phase commit). For information about access between different release levels of DB2 and different levels of DRDA, see Section 5 of *Application Programming and SQL Guide*.

  Using DB2 private protocol access, you *cannot*:

  – Create or drop objects in a remote system
  – Grant or revoke privileges or authorities in a remote system

  With either DB2 private protocol access or DRDA access, your application *cannot*:

  – Access more than one system in a single SQL statement
  – Define referential constraints between different systems

- Performance considerations

  With distributed data, an obvious consideration for an SQL query transmitted to a remote system is that the query and its reply must both be transmitted over a network. This is complicated even further when you add the ability to update at multiple sites—additional messages need to be sent among the participants in the unit of work. See “Considerations for Tuning Distributed Applications” on page 5-316 for more information about message overhead for distributed processing.

  The main benefit of stored procedures is that they reduce the number of times queries must be sent across a network by allowing the DB2 server to issue multiple SQL statements within a single stored procedure. This reduces an application's CPU costs and I/O delays, thus, improving end user elapsed time. Additionally, performance usually improves when using stored procedures because a DB2 client can use a stored procedure to execute multiple SQL statements at a DB2 server with a single network request. For more detailed information about stored procedures and performance, see Section 6 of *Application Programming and SQL Guide*.

With DB2 private protocol access, the need to bind SQL statements when they are first executed in a unit of work can increase processor overhead when compared with the same query executed on your local subsystem. Some applications, however, can improve performance using *block fetch*, which is described in “How Block Fetch Improves Performance” on page 5-318.

With DRDA access, you prebind your queries at the remote location, and execute the resulting package over and over again, so that additional time is not spent on binding the query for each unit of work. The package can contain PREPARE and EXECUTE statements, so you can send queries to remote locations using DRDA access. For more information about designing remote applications, see Section 5 of *Application Programming and SQL Guide*.

- Character conversion considerations.

  If the systems represent characters using different coded character sets, such as EBCDIC and ASCII, the DB2 SYSIBM.SYSSTRINGS catalog table must include a row that describes conversions from one coded character set to the other. Refer to Appendix B of *Installation Guide* for further information.

Implications for System Operations

If you administer a DB2 subsystem that has the distributed data facility (DDF), you can stop and start DDF without stopping DB2. For information about the DB2 commands for those operations and other associated ones, see “ Section 4. Operation and Recovery” on page 4-1.

You cannot, however, effectively administer a remote system from your local system. Hence, you must sometimes coordinate recovery operations on the two systems by some means outside of DB2. “Considerations for Recovering Distributed Data” on page 4-124 describes those considerations.

Also, each system controls access to its own data. “Chapter 3-2. Controlling Access to DB2 Objects” on page 3-13 tells how to grant privileges to users on remote systems.

DB2 traces provide ways to identify accesses from other systems. The resource limit facility (the governor) allows you to limit the time allotted to dynamic accesses and remote binds from other systems. See “Resource Limit Facility (Governor)” on page 5-76 for more information about the governor. See “DB2 Trace” on page X-177 for more information about traces.

DataHub can simplify administration of remote systems because it allows you to work from a single interface. For more information, see *IBM DataHub General Information*.

Stored Procedures

DB2's stored procedures function offers significant time and overhead savings for applications using DRDA. Stored procedures can be also be accessed locally. Using a stored procedure, the client application issues a single network operation to run the stored procedure. The stored procedure can issue many SQL statements using a single thread—the same thread used by the calling application—and return a parameter list in another single operation.

A stored procedure is simply a DB2 application program, containing SQL statements, that runs in a DB2-managed or WLM-managed stored procedures address space. With a single operation, a series of SQL statements are executed in the stored procedure, thus significantly decreasing the costs of distributed SQL statement processing.

For more information on stored procedures, see Section 6 of *Application Programming and SQL Guide* .

Chapter 2-9. Implementing Your Design

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

After you design a relational database, you must implement the design by allocating storage for the database and creating DB2 objects. This chapter includes physical design considerations and implementation information. The following topics are discussed:

- **Auxiliary storage**. All DB2 data is stored in VSAM data sets. You can let DB2 create and manage these data sets (using DB2 storage groups) or you can do it yourself (using VSAM access method services). This chapter explains what to do for either choice.
- **How to create DB2 objects**. This is described under:

  “Implementing Your Storage Groups” on page 2-82
  “Implementing Your Databases” on page 2-85
  “Implementing Your Table Spaces” on page 2-86
  “Implementing Your Tables” on page 2-92
  “Implementing Your Indexes” on page 2-99
  “Implementing Your Views” on page 2-105.

  If you are using DB2 to enforce referential integrity, see also “Chapter 2-3. Maintaining Data Integrity” on page 2-19.

  The natures and uses of DB2 objects are introduced in “Chapter 2-1. Designing a Database” on page 2-5. Information about the privilege or authority needed to create an object is in “Chapter 3-2. Controlling Access to DB2 Objects” on page 3-13.
- **What the DB2 catalog records**. This tells you how to retrieve information about DB2 objects. See “Chapter 2-11. Using the Catalog in Database Design” on page 2-117

You create objects by executing SQL statements. This chapter does not include either the complete syntax or a full description of all the effects of each SQL statement. For that information, refer to *SQL Reference*. This chapter also does not include information about creating objects to contain shared read-only data. See “Appendix F. Sharing Read-Only Data” on page X-153 for information about doing that.

Choosing Names for DB2 Objects

The allowable format for DB2 object names varies depending on what type of object is being named. See Chapter 3 of *SQL Reference* for more information about naming conventions.

DB2 Storage Groups and Databases

A name for either of these objects 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
Storage group	DSN8G510
Database	DSN8D51A

Table Spaces

A table space name is an identifier of up to eight characters, qualified by a database name. If you omit the database name, DB2 assumes the default database, DSNDB04. A typical name is:

Object	Name
Table Space	DSN8D51A.DSN8S51D

Tables, Views, and Indexes

A name for any of these objects is an identifier of up to 18 characters qualified by a short identifier of up to eight characters. Do not use SYSIBM as a qualifier in the names of tables, views, or indexes because it is used only for IBM-supplied objects.

For indexes, the index space name is an eight-character name. For DB2-managed data sets, DB2 generates a unique index space name. For user-managed data sets, DB2 uses the index name specified on the CREATE INDEX statement for the eight-character index space name. However, if the index name on the CREATE INDEX statement is greater then eight characters, DB2 uses the first eight bytes of that name. This index space name must be unique among all index space and table space names in the database.

Names of some objects are:

Object	Name
Table	DSN8510.EMP
View	DSN8510.VDEPMG1
Index	DSN8510.XACTYPE1
DB2 catalog table	SYSIBM.SYSINDEXES

Naming Remote Objects for DB2 Private Protocol Access

Unlike DRDA access, in which you use the SQL CONNECT statement to direct activity to a certain database management system, DB2 private protocol access relies on location names to refer to objects from other DB2 subsystems. Therefore, DB2 supports the use of names with three parts for tables and views.

Suppose that data is occasionally moved from one DB2 subsystem to another. Users who query that data want those moves to be transparent. They want to log on always to the same system and access the same table or view, no matter where the data is. They do that by using *aliases*.

Three-Part Names for DB2 Private Protocol Access

You can access a table or view at a specific DB2 subsystem using DB2 private protocol access by using a name containing the following three parts, separated by periods:

- **A LOCATION name**: This is the name that appears in the LOCATION column of table SYSIBM.LOCATIONS in the communications database (CDB) and uniquely identifies the subsystem you are addressing within the VTAM network. The LOCATION name is the same as the DRDA RDB_NAME used by non-DB2 database management systems using DRDA.
- **An AUTHORIZATION ID**: This identifies the owner of the table or view at the location you are addressing.
- **An OBJECT name**: This is the name of the table or view at the location you are addressing.

For example, if the SYSIBM.LOCATIONS table at your local subsystem contains an entry for location USIBMSTODB21, you can access the sample employee table at USIBMSTODB21 in a query like this:

```
SELECT SUM(SALARY), SUM(BONUS), SUM(COMM)
  FROM USIBMSTODB21.DSN8510.EMP;
```

It is also possible to use names with three parts for local tables or views.

With few exceptions, a name with three parts can be used wherever a table or view can be used. The exceptions are:

- The LOAD and RUNSTATS utilities
- The CREATE SYNONYM statement.

Aliases for DB2 Private Protocol Access

An alias is a substitute for the three-part name of a table or view. The alias can be up to 18 characters long, qualified by an owner ID.

You can use an alias wherever you can use a table or a view, with the following exceptions:

- You cannot use an alias in the LOAD and RUNSTATS utilities.
- You cannot create a synonym for an alias that refers to an object from a remote DB2 subsystem. (However, you can create a synonym for an alias that refers to a local table or a view.)
- You cannot create a view on an alias that refers to an object from a remote DB2 subsystem. (However, you can create a view on an alias that refers to a local table or a view.)

The CREATE ALIAS Statement: Use the SQL CREATE ALIAS and DROP ALIAS statements to manage aliases. An example of a CREATE ALIAS statement follows.

```
CREATE ALIAS TESTTAB FOR USIBMSTODB22.DSN8510.EMP;
```

If the user with the ID JONES dynamically creates the alias, then JONES owns the alias, and you query the table like this:

```
SELECT SUM(SALARY), SUM(BONUS), SUM(COMM)
  FROM JONES.TESTTAB;
```

The object for which you are defining an alias does not have to exist when you execute the CREATE ALIAS statement. However, the object must exist when a statement that refers to the alias is executed.

Who Can Use Aliases

After an alias has been created, it can be used by anyone who has been granted authority to the object the alias is referencing. A user does not need to be granted a separate privilege to use the alias.

Comparing Three-Part Names and Aliases

When you use DB2 private protocol access, you can always use three-part names to reference data at another DB2 subsystem. The advantage of three-part names is that they allow application code to be executed at different DB2 locations without the additional overhead of alias maintenance. However, if the table locations change, then the applications affected must be changed.

The advantage of aliases is that aliases allow you to move data around without having to modify application code or interactive queries. However, if you move a table or view, you must drop aliases that refer to those tables or views, then re-create them with the new location names.

The relative costs and benefits of aliases are summarized in Table 12.

Table 12. Relative Costs and Benefits of Three-part Names vs. Aliases

Situation	Three-part names	Aliases
A table is moved from one location to another.	Three-part names in applications referring to the moved table must be changed.	The alias for that table must be dropped, then re-created with the new location name.
An application is moved from one location to another.	No change.	Aliases must exist at new location for all tables referred to by the application. Aliases at the old location can be dropped if they are no longer needed.

For more information about using aliases, see Chapter 3 of *SQL Reference* .

Implementing Your Storage Groups

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 instance) to fast devices and seldom-used tables to slower devices; that choice of 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 have been created, you can process them with access method service commands that support VSAM control-interval (CI) processing (for example, IMPORT and EXPORT).

- When a table space is dropped, DB2 automatically deletes the associated data sets.
- When a data set in a 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 5-100.
- When creating or reorganizing a table space, if the associated data sets already exist, DB2 deletes and then redefines them.
- When you want to move data sets to a new volume, you can alter the volumes list in your storage group. DB2 will automatically relocate your data sets during utility operations that build or rebuild a data set (LOAD REPLACE, REORG, and RECOVER). With user-defined data sets, on the other hand, you must delete and redefine your data sets in order to move them.

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. “Chapter 2-11. Using the Catalog in Database Design” on page 2-117 provides more information.

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. We describe the alternatives under CREATE STOGROUP Statement below, and “Managing Your Own DB2 Data Sets” on page 2-68. Yet another possibility is to have SMS manage some or all of your DB2 data sets. For information about this, see “Managing Your DB2 Data Sets with DFSMShsm” on page 2-67.

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. You must identify the integrated catalog facility catalog (the "integrated catalog") when you create a storage group or when you create a table space that does not use storage groups.

CREATE STOGROUP Statement

To create a DB2 storage group, use the SQL statement CREATE STOGROUP. This This statement provides a list of volumes that DB2 can use. For detailed information on CREATE STOGROUP, see Chapter 6 of *SQL Reference*.

When DB2 processes the CREATE STOGROUP statement, it does not check the existence of the volumes in the VOLUMES clause or determine the types of devices they identify. Later, when the storage group is used to allocate data sets, DB2 passes the list of volumes in the order you specified to Data Facilities (DFSMSdfp), which does the data set allocation.

All the volumes in a storage group must be of the same type. If the volumes you name are not mounted, or are not all of the same device type, a dynamic allocation error occurs when you try to create a table space or index.

A dynamic allocation error can also occur when trying to extend the table space or index if any of the following are true:

- The volumes are of different device types
- Any volid in the STOGROUP is varied offline
- Any volid in the STOGROUP is invalid

Below are two examples; the sections that follow explain their principal elements.

```
CREATE STOGROUP DSN8G510
  VOLUMES (DSNV01)
  VCAT DSNC510
  PASSWORD DSNDEFPW;

CREATE STOGROUP G201
  VOLUMES ("33333", "44444")
  VCAT CAT2
  PASSWORD XB17R;
```

Names

The statements above create two STORAGE GROUPS, DSN8G510 and G201. (DSN8G510 is the storage group for the DB2 sample application.) You cannot have two DB2 storage groups with the same name in the same DB2 system. Other objects, however, can have the same name as a DB2 storage group.

VOLUMES Clause

Storage group DSN8G510 is assigned one volume and G201 is assigned two. A given volume can belong to more than one DB2 storage group.

You can add or delete volumes using the ALTER STOGROUP statement. The changes do not affect the storage of existing objects in the storage groups, but do affect the future assignment of storage within the group; for example, an object you define later can be assigned to an added volume, but cannot be assigned to a deleted volume.

To allow SMS to manage your storage groups, you can use asterisks (nonspecific volume IDs) in the VOLUMES clause. However, do not mix specific and nonspecific volume IDs, whether in a new or existing storage group. For information about using SMS to manage your storage groups, see “Managing Your DB2 Data Sets with DFSMShsm” on page 2-67. If you are using shared read-only data, see “Create DB2 Storage Groups” on page X-158 for additional storage management considerations.

VCAT clause

Identifies the integrated catalog in the VCAT clause.

PASSWORD Clause

If your integrated catalog facility catalog is password protected, then when you create a DB2 storage group, you must supply the control or master integrated catalog password in the PASSWORD clause. The password does not apply to data sets managed by DFSMS/MVS's storage management subsystem (SMS). Data sets defined to SMS should be protected by RACF or some similar external security system.

Default Storage Group

There is a *system default storage group*, SYSDEFLT, 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.

Implementing Your Databases

When you define a DB2 database, you name an eventual collection of tables and associated indexes, as well as the table spaces in which they reside. In deciding whether to define a new database for a new set of objects, consider the following factors:

- 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, it is often convenient to place a set of tables that are used together into the same database. (The same database will hold any 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, it might be inconvenient to place many unrelated tables in a single database.

  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, we recommend that each QMF user have a separate database.
- It is possible for internal database descriptors (DBDs) to become inconveniently large; Section 2 of *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 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. Thus, limiting the size of DBDs is another reason to define new databases. The MODIFY utility is described in Section 2 of *Utility Guide and Reference*.

CREATE DATABASE Statement

To create a database, use the CREATE DATABASE statement. For detailed information on CREATE DATABASE, see Chapter 6 of *SQL Reference*.

The example below shows the SQL statement that can be used to create the sample database DSN8D51A:

```
CREATE DATABASE DSN8D51A
  STOGROUP DSN8G510
  BUFFERPOOL BP0;
```

STOGROUP clause

This clause establishes a default DB2 storage group, DSN8G510, for the database. If you do not name a storage group, the default storage group for the database is SYSDEFLT.

BUFFERPOOL clause

This clause establishes BP0 as the default buffer pool for table spaces and indexes within the database.

You can also use the STOGROUP and BUFFERPOOL clauses in the CREATE TABLESPACE and CREATE INDEX statements. If you use them in the CREATE DATABASE statement, you do not have to use them each time you create a table space or index. If you do use them again in CREATE TABLESPACE and CREATE INDEX, you override the defaults you established with CREATE DATABASE. Furthermore, you might specify that you are not using a DB2 storage group, or that you are using the default storage group of the database, but with a new primary or secondary space allocation.

CCSID Clause

This clause specifies the default encoding scheme, ASCII or EBCDIC, for data stored in this database. This default applies to table spaces created in this database. All tables stored within a table space must use the same encoding scheme. This option defaults to the default encoding scheme that was specified during installation.

Using the Default Database

You do not have to define a database to use DB2; you can use the default database (DSNDB04). This means that you can define tables and indexes without specifically having to define a database. The catalog table SYSIBM.SYSDATABASE describes all databases, including the default.

Implementing Your Table Spaces

Table spaces are the physical spaces that hold tables. A table space can have one or more tables. Each table space can hold a maximum of 64 GB of data and might use one or more VSAM data sets. Table spaces are divided into units called pages that are either 4KB or 32KB in size. As a general rule, you should have no more than 10 or 20 table spaces in one DB2 database. See the information under “Implementing Your Table Spaces” for a discussion of the factors that influence this recommendation.

It is possible to compress data in a table space, which can allow you to store more data per data page. For more information, see “Compressing Data in a Table Space or Partition” on page 2-63.

Creating a Table Space Implicitly

As with DB2 storage groups and databases, it is not necessary to create a table space before you create a table unless you are managing all your own data sets. When you use CREATE TABLE, DB2 generates a table space for you. However, DB2 will generate a table space only if you use CREATE TABLE without specifying an existing table space name. If you do not name 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 8 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 4 characters of the table name, and *nyyy* is a digit and 3 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.

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. You can create simple, segmented, and partitioned table spaces.

CREATE TABLESPACE Statement

This section explains the various clauses of the CREATE TABLESPACE statement. For detailed information on CREATE TABLESPACE, see Chapter 6 of *SQL Reference* .

IN Clause

If you want the table space to be part of a specific database, use the name of that database in this clause. The named database must have been previously defined. If you do not name a database, DB2 uses the default database DSNDB04.

USING Clause

If you do not include the USING clause in your CREATE TABLESPACE statement, DB2 uses the default storage group of the database.

- STOGROUP or VCAT

  Use STOGROUP and the name of a DB2 storage group to create the table space in a specific DB2 storage group. Use VCAT and the name of an integrated catalog facility catalog to create the table space in VSAM data sets that you have defined yourself.

- PRIQTY and SECQTY with STOGROUP

  If you use STOGROUP, the PRIQTY and SECQTY clauses allow you to specify the primary and secondary space allocations for the table space. The clauses are optional, but consider using them. You specify the allocation as a number of kilobytes. The default value of PRIQTY is three pages. For large tables, you need a larger allocation. For nonpartitioned table spaces, do not use a value larger than 2 GB, because DB2 allocates a new data set when that limit is reached, leaving the rest of the allocation unused. For partitioned table spaces, do not set PRIQTY to a value greater than the partition size. The default value

of SECQTY is 10 percent of the primary quantity or three times the page size, whichever is larger.

For more information about how space allocation can affect the performance of mass inserts, see “Ensure Allocation in Cylinders” on page 5-40.

If you use secondary allocations, they are probably not located either next to the primary allocation or next to each other. Therefore, if an application accesses data from both areas, performance might suffer. If possible, use a value for PRIQTY that minimizes the need for DB2 to go to secondary allocations, without wasting space. DB2 can extend a simple or segmented table space to more than one data set, but only if the value of (PRIQTY + 118 × SECQTY) is at least 2 GB.

You might possibly get up to 123 extents before reaching the limit, but 119 is more likely.

When DB2 finds that the space available for your table space is getting low, it gives you a warning message. For information about what to do when you are running out of space, see “Out of DASD Space or Extent Limit Reached” on page 4-188.

The amount of space allocated to table spaces is recorded in the SYSIBM.SYSTABLESPACE table of the DB2 catalog by the STOSPACE utility. STOSPACE is described further in “Appendix G. Using Tools to Monitor Performance” on page X-173.

You can change the values of PRIQTY, SECQTY, and ERASE by altering the table space definition. For instructions, see “Altering Table Spaces” on page 2-125.

- ERASE with STOGROUP

  The ERASE clause determines what action VSAM takes when a DB2 storage group-defined table space is dropped, either through the DROP TABLESPACE or DROP DATABASE SQL statements, or from the LOAD REPLACE, REORG, or RECOVER utilities. If you use ERASE YES, the data is overwritten with zeros, as a security measure. If you use ERASE NO, the data is not necessarily overwritten. For more information about what happens with ERASE, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

  Using the default, ERASE NO, means that deletion happens more quickly. With ERASE NO, the dropped data is still accessible in machine-readable form, though not through DB2.

BUFFERPOOL Clause

Your choice of buffer pool name implicitly determines the page size you get—buffer pools BP0 through BP49 result in 4KB page sizes, and BP32K through BP32K9 result in a page size of 32KB. The choice of buffer pool is not critical at this stage; you can change it easily with the ALTER TABLESPACE statement.

In most cases, you can omit the clause and use the buffer pool of the database by default. Make sure the buffer pool you specify is defined with a nonzero size and that you are authorized to use that buffer pool. Otherwise, you get an error when you try to create or alter the table space.

For more information about buffer pools, see “Tuning Database Buffer Pools” on page 5-49.

LOCKSIZE Clause

In most cases, you can probably omit this value, and use LOCKSIZE ANY by default. That choice leaves the decisions about the lock size to DB2, and the system will usually choose LOCKSIZE PAGE and LOCKMAX SYSTEM (see below). When LOCKSIZE TABLESPACE is specified, LOCKMAX must be 0. Before you choose LOCKSIZE TABLESPACE you should know why you do not need concurrent access to the object. See “Other Options that Affect Locking” on page 5-168 for information about the effects of the LOCKSIZE clause.

LOCKMAX Clause

This specifies the maximum number of page or row locks an application process can hold simultaneously in the table space. If a program requests more than that number, locks are escalated. The page or row locks are released and the intent lock on the table space or segmented table is promoted to S or X mode.

You use an integer to specify the number of locks allowed before escalating, or you can specify SYSTEM, which uses the value set as LOCKS PER TABLESPACE on installation panel DSNTIPJ.

CLOSE Clause

The CLOSE clause tells the priority DB2 should use when determining which open data sets to close. DB2 defers deallocating the table space's data sets until the number of open data sets reaches either a maximum determined by certain installation parameters (the derived value DSMAX) or your MVS limit. If this maximum is reached, the least recently used table spaces that were defined with CLOSE YES are closed first.

CLOSE NO table spaces are only closed if there are not enough CLOSE YES table spaces to relieve the excess number of open data sets. For most table spaces, CLOSE YES is the recommended option. For more information about the CLOSE clause, see “Understanding the CLOSE YES and CLOSE NO Options” on page 5-90.

DSETPASS Clause

If you are using DB2 storage groups, the password you give is the one that protects the data sets and the one that is passed to VSAM when the data sets are used by DB2. If you are defining the data sets yourself, the password you give is only the password that is passed to VSAM when the data sets are used by DB2. In the latter case, the password that protects the data sets is defined by you, using VSAM access method services. All password-protected data sets of the table space must have the same password.

PCTFREE and FREEPAGE Clauses

PCTFREE tells you the percentage of each page that is to be reserved as free space during LOAD and REORG operations. The space can be used by later inserts or updates. The default value is 5.

If you can foresee adding a new column to a table, you might want to provide additional free space in the table space. Otherwise, inserting values in the new column is likely to force rows out of sequence. For more information about choosing values for PCTFREE and FREEPAGE, see “Reserve Free Space in Table Spaces and Indexes” on page 5-38.

COMPRESS Clause

Use COMPRESS YES to compress data in a table space or in any or all partitions in a partitioned table space. The corresponding index data is *not* compressed. Catalog, directory, and work file table spaces are not compressed, either.

The data is not compressed until a dictionary is built for the table space or partition, by using the LOAD utility (with REPLACE or RESUME NO) or the REORG utility. After a dictionary is built, any rows inserted or updated are compressed if compression reduces row length.

Although data compression can reduce the amount of DASD space required to store data, it can also require a significant amount of processor overhead. Before deciding to compress, see "Compressing Data in a Table Space or Partition" on page 2-63 for more information.

CCSID Clause

This clause specifies the encoding scheme, ASCII or EBCDIC, of tables stored in this table space. The CCSID option defaults to the encoding scheme of the database in which this table resides.

MAXROWS Clause

Use MAXROWS *integer* to specify the number of rows that DB2 will consider placing on each page, where the *integer* can range from 1 to 255. If MAXROWS is not specified, the default number of rows is 255. For more information, see " Section 5. Performance Monitoring and Tuning" on page 5-1.

Creating a Segmented Table Space

Figure 41 shows an example of a CREATE TABLESPACE statement that creates a segmented table space.

```
CREATE TABLESPACE DSN8S51C
  IN DSN8D51P
  USING STOGROUP DSN8G510
        PRIQTY 30720
        SECQTY 10240
  SEGSIZE 32
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE YES
DSETPASS DSN8;
```

Figure 41. Example of CREATE TABLESPACE, for a Segmented Table Space

SEGSIZE Clause

The value given is the number of pages in each segment; it must be a multiple of 4, from 4 to 64. The choice of the value depends on the size of the tables to be stored. For efficient use of sequential prefetch of tables that are smaller than 32 pages and that do not grow, choose a segment size close to the number of pages in the largest table in the table space. However, if there are many tables that take less space than an entire segment, then space is wasted in each segment.

But if most of the tables require several segments, then access can be made more efficient by choosing a larger segment size such as 32 or 64. For information about choosing a value for SEGSIZE, see “Table Space Scans (ACCESSTYPE=R PREFETCH=S)” on page 5-275.

LOCKSIZE Clause

The TABLE option is valid only for segmented table spaces. It means that DB2 can acquire locks that lock a single table, rather than the entire table space. See “Other Options that Affect Locking” on page 5-168 for information about the effects of the LOCKSIZE clause.

FREEPAGE Clause

If you want to leave full pages of free space in a segmented table space, you must have at least one free page in each segment. A value of FREEPAGE that is larger than the value of SEGSIZE makes no sense because it means that more than a segment should be filled before leaving a page free. Therefore, if you use a FREEPAGE value larger than SEGSIZE, it is effectively reduced to (SEGSIZE -1). For example, if you use FREEPAGE 30 with SEGSIZE 20, the value of FREEPAGE is interpreted as 19, and you get one free page in each segment.

Creating a Partitioned Table Space

A partitioned table space can have only one table, and up to 64 partitions. The maximum size of any table space is 64 GB, but the maximum size of a partition is 4 GB; therefore, a table space with fewer than 16 partitions cannot reach the maximum size. Each partition can be thought of as a unit of storage. DB2 creates and manages a separate VSAM data set for each partition. If you are managing your own VSAM data sets, you must do the same, or you can allocate space for each partition using the PART option of the NUMPARTS clause.

Creating a Large Partitioned Table Space

You may also consider creating large partitioned table spaces which support tables with more than 64 GB of data, or more than 64 partitions. With large partitioned table spaces, you can have up to 254 partitions of 4 GB each, while a *non-large* partitioned table space can have up to 64 partitions. This allows the size of partitioned tables to increase to roughly one terabyte. In addition, each index partition of the partitioned index is 4 GB. The maximum size of a data set for a nonpartitioned index on a LARGE partitioned table space is 4 GB. With a limit of 128 datasets, the maximum size of a nonpartitioned index is 512 GB.

Figure 42 on page 2-92 creates a large table space, SALESHX. The first USING clause establishes the default DB2 storage group and space allocations for all partitions. The example assumes that this table space is used by a large query database application to record historical sales data for marketing statistics.

```
CREATE LARGE TABLESPACE SALESHX
  IN DSN8D51A
  USING STOGROUP DSN8G510
    PRIQTY 4000
    SECQTY 130
    ERASE NO
NUMPARTS 82
 (PART 52 USING STOGROUP DSN8G510
    PRIQTY 4000
    SECQTY 130
    COMPRESS YES,
  PART 75 USING STOGROUP DSN8G510
    PRIQTY 4000
    SECQTY 130
    COMPRESS NO)
LOCKSIZE PAGE
BUFFERPOOL BP1
CLOSE NO;
```

Figure 42. Example of CREATE LARGE TABLESPACE, for a Partitioned Table Space

The CREATE TABLESPACE statement in Figure 42 creates a table space with 82 partitions by using the NUMPARTS clause. The statement illustrates how the partitions can be stored on different device types; partition 52 uses storage group DSN8G510 and has compressed data.

Implementing Your Tables

Designing tables to be used by many applications is a critical task. Table design can be difficult because the same information can be represented many ways. In addition, decisions between the conflicting objectives of logical design and a physical design must be made. An example of such a conflict is normalization, described in “Normalize Your Tables to Avoid Redundancy” on page 2-13. Later, you may have to make changes to your tables. The ALTER TABLE statement lets you make changes such as adding columns, add or drop a primary or foreign key, add or drop table check constraints, change the AUDIT, VALIDPROC or DATA CAPTURE clauses. Changes in design should be carefully considered to avoid or reduce the disruption of your applications.

If you have DBADM authority, you probably want to control the definition of DB2 databases and table spaces because of their impact on the performance, storage, and security of the entire relational database. In most cases, you also want to keep the responsibility for creating tables. After designing the relational database, create the necessary tables for application programs, then pass the authorization for their use to the application developers—either directly or by using views.

But if you want to, you can grant the authority for creating tables to those responsible for implementing the application. For example, you probably want to authorize certain application programmers to create tables if they need temporary tables for testing purposes.

If some users in your organization want to use DB2 with minimum assistance or control, you can define a separate storage group and database for these users and authorize them to create whatever data objects they need, including tables.

Table Names

For both DB2 base tables and temporary tables, a table name can have up to 18 characters. Table names that are not explicitly qualified by you are implicitly qualified by DB2. For example, assume that someone with an SQL ID of SMITH is at a terminal entering dynamic SQL statements. If SMITH creates a table named ABC, with no qualifier, DB2 uses SMITH as the qualifier and considers that the table name is SMITH.ABC. SMITH cannot own another table, view, or alias called ABC. A different SQL ID (say, JONES) can create another table, view, or alias called ABC. DB2 recognizes the second table as JONES.ABC.

CREATE TABLE Statement

To create a table you have designed, use the CREATE TABLE statement. When you create a table, DB2 records a definition of the table in the DB2 catalog. Creating a table does not store the application data. You can put data into the table by several means, all described in “Chapter 2-10. Loading Data into DB2 Tables” on page 2-113.

Figure 43 shows the statement used to create the sample employee table.

```
CREATE TABLE DSN8510.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,
       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 DSN8510.DEPT
               ON DELETE SET NULL        )
  EDITPROC  DSN8EAE1
  IN DSN8D51A.DSN8S51E;
```

Figure 43. Example of CREATE TABLE

Further information about the clauses of the CREATE TABLE statement can be found in other sections of *Administration Guide* and in Chapter 6 of *SQL Reference*.

Clauses of the CREATE TABLE Statement

This section explains the various clauses of the CREATE TABLE statement. However, the clauses discussed here for the CREATE TABLE statement do NOT apply to the CREATE GLOBAL TEMPORARY TABLE statement with the exception of the CCSID clause.

PRIMARY KEY Clause

This clause defines a primary key composed of specified columns. You can have only one primary key per table and the specified columns must be defined as NOT NULL. Each column name must be an unqualified name that identifies a column of the table and you can specify each column only once. The number of specified columns must not exceed 64, and the sum of their length attributes must not exceed 255.

The following example shows how to define a composite primary key.

```
CREATE TABLE DSN8510.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) )
  IN DATABASE DSN8D51A;
```

Figure 44. Defining a Composite Primary Key

(This example omits the clauses that define the foreign keys in the table.)

FOREIGN KEY Clause

A foreign key of a table is a key that is specified in the definition of a referential constraint. The foreign key must have the same number of columns, with the same descriptions, as the parent key of the parent table. For information on how to define foreign keys, refer to “Defining a Foreign Key” on page 2-22.

The following example shows how to define a list of columns as a foreign key of a table.

```
CREATE TABLE DSN8510.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 DSN8510.PROJACT
               ON DELETE RESTRICT,
       FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8510.EMP
               ON DELETE RESTRICT         )
  IN DATABASE DSN8D51A;
```

Figure 45. Creating the Employee to Project Activity Table.

Rules for the FOREIGN KEY Clause

Keep the following rules in mind about the FOREIGN KEY clause:

- The order of the foreign key columns must match the order of the primary key columns of the parent table. The descriptions of the foreign and primary key columns must match, except for the names, default values, and null attributes.
- The referred to (parent) table must already exist and have a complete definition. The parent cannot be a table in the DB2 catalog. The parent table does not have to be in the same DB2 database or table space as the table in which you define a foreign key.
- A foreign key cannot reference a view.
- The clause defines a relationship, naming one of the delete rules from the list in “DELETE Rules” on page 2-27.
- If multiple pathways exist between tables T1 and T2 in which T1 is a parent of T2 or T1 has a CASCADE relationship to a parent of T2, then *the relationships that enter T2 must be the same and must not be SET NULL*.

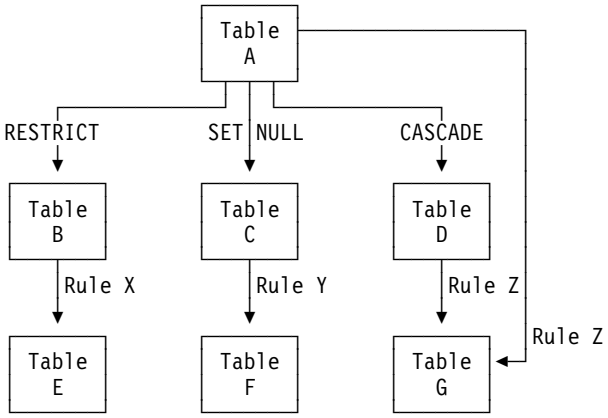


Figure 46. Sample Pathways Between Tables

For example, in Figure 46, a delete operation on table A enters G through two different pathways because G is both a dependent of A and a dependent of D, to which deletes from A cascade. Therefore, Rule Z must be the same in both places, either both RESTRICT, or both CASCADE, or both NO ACTION. There is no restriction on delete rules X and Y, because E and F are not affected by a delete operation on table A.

This restriction exists because DB2 does not allow you to create ambiguous situations in which the result of a delete operation would depend on the order in which the deletions occur.

- A table cannot have two foreign keys on the same list of columns that reference the same parent table. The table can have more than one foreign key, as the sample project table has, and two or more foreign keys (on different lists of columns) can reference the same parent table. Also, two foreign keys can use the same list of columns, referring to different parent tables.
- In a distributed system, the parent table cannot reside in a different subsystem from its dependents.

Authorization: Creating a table with a foreign key requires the usual privileges for the dependent table. In addition, the parent table must have the ALTER privilege, or the columns of the parent key must have the REFERENCES privilege.

CHECK Clause

This clause defines a table check constraint. A *check-condition* is a search condition, with the following restrictions:

- It can refer only to columns of table *table-name*.
- It can be up to 3800 bytes long, not including redundant blanks.
- It must not contain any of the following:
  - Subselects
  - Functions
  - Host variables
  - Parameter markers
  - Special registers
  - Columns that include a field procedure
  - Quantified predicates
  - EXISTS predicates

EDITPROC and VALIDPROC Clauses

These clauses allow you to specify two exit routines, an edit routine and a validation routine, which deal with entire rows rather than the values of a single column. “Edit Routines” on page X-44 and “Validation Routines” on page X-48 describe these routines and explain how to use them.

IBM supplies two sample edit routines: One is called DSN8EAE1 and encodes and decodes the data for the salary column of the employee sample table. The other sample compresses data by the Huffman algorithm and is called DSN8HUFF. Before using any data compression routine, understand its limitations and consider tailoring it to your particular table. For the restrictions and concerns that apply to the IBM sample, see the comments provided with the code. Both samples reside in library *prefix*.SDSNSAMP.

DB2 provides a more efficient method for compressing data than DSN8HUFF. For more information, see “Compressing Data in a Table Space or Partition” on page 2-63.

FIELDPROC Clause

A field procedure is often used to change the sorting sequence of the values in a short string column. To specify that a column uses a field procedure, use the option FIELDPROC, followed by the application program name of the procedure and, optionally, a list of parameters. For example, to specify a field procedure for the column LASTNAME of the employee sample table, we changed one line of Figure 43 on page 2-93 to look like this:

```
LASTNAME  VARCHAR(15) NOT NULL FIELDPROC MYPROG (4, 3, 7),
```

In the example, the name of the field procedure is chosen as “MYPROG.” For instructions about writing a field procedure, see “Field Procedures” on page X-57 .

IN Clause

Use the IN clause of CREATE TABLE to name the DB2 database or the database and the table space to be used for the table. If you do not name a database, DB2 creates the table in the default database DSNDB04. If you do not name a table space, DB2 creates a simple table space for you as described in “Creating a Table Space Implicitly” on page 2-86. You might also want to consider whether the table warrants a new database; some pertinent factors are described for IN Clause of “CREATE TABLESPACE Statement” on page 2-87.

UNIQUE Clause

If UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

The NOT NULL clause must be specified with this clause. This clause cannot be specified more than once in a column definition. Columns in a UNIQUE clause cannot be parent keys.

AUDIT Clause

The audit trace allows you to list events of various types that occur in audited tables. The AUDIT clause tells whether a particular table is audited. It has three options: ALL, CHANGES, and NONE. These options are described in more detail in “Auditing a Specific Table” on page 3-123.

For an exact description of the events that are audited for each trace class and instructions on using the trace, see “Options of the Audit Trace” on page 3-120. There is likely to be a performance cost in auditing accesses of any type, so consider carefully how you intend to use the trace before including an AUDIT clause in the table definition.

DATA CAPTURE Clause

Use this clause when you want data changes to this table to be written to the log in an expanded format that can be retrieved by such programs as the Log Apply Feature of the Remote Recovery Data Facility (RRDF) program offering or Data Propagator Relational (DPropR).

OBID Clause

Use this clause when you want to use a specific object identifier or when you are re-creating table definitions for shared read-only data. For more information about shared read-only data, see “Appendix F. Sharing Read-Only Data” on page X-153.

WITH RESTRICT ON DROP Clause

If a table is defined with WITH RESTRICT ON DROP, it can be dropped only after it is altered to remove the restriction. Also, a table space or database that contains a table with WITH RESTRICT ON DROP cannot be dropped until the table is altered to remove the defined restriction.

CCSID Clause

Use the CCSID clause to specify the encoding scheme, ASCII or EBCDIC, for data stored in the table. If an IN clause is specified with a table space name then the value must agree with the encoding scheme already in use for that table space. All data stored within a table space must use the same encoding scheme.

If an IN clause is not specified with a table space name then a table space is created implicitly and the encoding scheme of that table space is the same as the table being created.

The CCSID option defaults to the encoding scheme of the table space containing this table. In addition, the CCSID option defaults to the DEFAULT ENCODING SCHEME install option for a table created in an implicit table space.

CREATE GLOBAL TEMPORARY TABLE Statement

DB2 Version 5 introduces the CREATE GLOBAL TEMPORARY TABLE statement allowing the creation and use of temporary tables which have a subset of the attributes of DB2 base tables. The term “global” for DB2 temporary tables means global to an application's process at the current server.

When you need a table only for the life of an application process, you can create a temporary table. DB2 does not log or lock temporary tables, so SQL statements that use them can be processed more efficiently. Temporary tables are especially useful when you need to sort or query intermediate result sets that contain large numbers of rows, but you want to store only a small subset of those rows permanently.

Temporary tables also have a number of uses for stored procedures. For more information, see Section 6 of *Application Programming and SQL Guide*.

Figure 47 shows the statement used to create a definition of a temporary table called TEMPPROD:

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD
  (SERIAL       CHAR(8)     NOT NULL,
   DESCRIPTION  VARCHAR(60) NOT NULL,
   MFGCOST      DECIMAL(8,2),
   MFGDEPT      CHAR(3),
   MARKUP       SMALLINT,
   SALESDEPT    CHAR(3),
   CURDATE      DATE        NOT NULL);
```

Figure 47. Example of CREATE GLOBAL TEMPORARY TABLE

CCSID Clause

Use the CCSID clause to specify the encoding scheme, ASCII or EBCDIC, for data stored in this table. This option defaults to the DEFAULT ENCODING SCHEME in the INSTALL panel for temporary tables.

Distinctions Between Base and Temporary Tables in DB2

Table 13 on page 2-99 summarizes important distinctions between DB2 base tables and DB2 temporary tables.

Table 13. Important distinctions between DB2 base tables and DB2 temporary tables.

Base Tables	Temporary Tables
CREATE TABLE statement puts description of table in catalog table SYSTABLES. That table description is a base table.	CREATE GLOBAL TEMPORARY TABLE statement puts description of table in catalog table SYSTABLES. That table description is a base table.
CREATE TABLE statement creates one empty instance of the table.	CREATE GLOBAL TEMPORARY TABLE statement does NOT create an instance of the table. An empty instance of a given temporary table is instantiated with the first implicit or explicit reference to the named temporary table in an OPEN, SELECT, INSERT, or DELETE operation executed by any program in the application process. In addition, an instance of the table is not a base table.
Any references to that table name in multiple application processes refer to a single base table at the current server.	References to that table name in multiple application processes refer to a distinct instantiation of the temporary table for each application process at the current server.
Table space and database operations, locking, logging, and recovery do apply.	Table space and database operations, locking, logging, and recovery do NOT apply.

A more detailed example of implementing temporary tables as well as restrictions and extensions of temporary tables can be found in Section 2 of *Application Programming and SQL Guide* and in Chapter 6 of *SQL Reference*. For information about temporary tables and their impact on DB2 resources, see “Work File Data Sets” on page 5-93.

Implementing Your Indexes

Indexes provide efficient access to data. This section describes clauses of the CREATE INDEX statement and gives an example of how to create an index for a large table. It also provides examples of creating indexes on partitioned and large partitioned table spaces.

CREATE INDEX Statement

Use the CREATE INDEX statement to create an index. If the table being indexed is empty, DB2 creates the index, but does not create index entries until the table is loaded or rows are inserted. If the table is not empty, you can choose whether to have DB2 build the index right away (when CREATE INDEX is executed) or you can defer the build of the index until later. Optimally, you should create all the indexes on a table before loading the table; however, if you have a populated table, choose the DEFER option and build the index using RECOVER INDEX.

The indexes you create have the same encoding scheme, ASCII or EBCDIC, as their associated table. Index entries for an ASCII table are stored in ASCII order. All indexes defined in an ASCII table must be type 2 indexes.

If you are creating an index for a table in a shared DB2 database, you need to be aware of the information in “Appendix F. Sharing Read-Only Data” on page X-153 before you create the index.

If you are creating an index for a large partitioned table space, See “Creating a Partitioned Index on a Large Partitioned Table Space” on page 2-103.

For nonpartitioned indexes, you can use the PIECESIZE clause to indicate how large DB2 should make the data sets that make up a nonpartitioned index. See “PIECESIZE Clause” on page 2-102 for more information.

Clauses of the CREATE INDEX Statement

This section describes the various clauses of the CREATE INDEX statement.

TYPE Clause

This allows you to specify the format of the index as either type 1 or type 2. Type 2 indexes are recommended. If you do not specify the TYPE option, the type is based on either the LOCKSIZE of the table space containing the table on which the index is being defined, or the default. For more information on default index type, see “Index Types and Recommendations” on page 2-51.

Type 1 indexes are not permitted on tables defined in a table space with a LOCKSIZE specification of ROW.

The default index type is specified during installation. See *Installation Guide* for more information.

UNIQUE and UNIQUE WHERE NOT NULL Clauses

This option specifies a constraint on the index key, preventing the table from containing rows that are duplicates with respect to the values of the columns. If the table already contains rows that are duplicate values of the columns, the index is not created.

If the column of the key allows null values and only UNIQUE is specified, null values are treated like any other values in the enforcement of the uniqueness constraint. For example, if the key is a single column, the index can contain no more than one null value. If any column of the key allows null values and UNIQUE WHERE NOT NULL is specified, the uniqueness constraint does not apply to a key value where any component is null.

The WHERE NOT NULL option is valid only for Type 2 indexes.

PCTFREE and FREEPAGE Clauses

These clauses enable you to specify an amount of space to be reserved when the index is created on a populated table, or when index entries are created or reorganized as a result of executing a DB2 utility. The reserved space is used later, when new index entries are created, to reduce the number of times a leaf page must be split to accommodate a new entry.

The clauses function much as they do in CREATE TABLESPACE:

- PCTFREE tells the percentage of each page that is to be reserved as free space when the index entries are created, either as the result of executing a DB2 utility or at the time the index is created. In leaf pages, the free space is divided equally among all subpages. For nonleaf pages, a maximum of 10 percent is left free. (If you specify a PCTFREE value of over 10, only 10 percent is left free.)

- FREEPAGE specifies how often DB2 is to leave a page of free space when the index entries are created.

In an index space that has many inserts, as in a simple table space, you might want to request the maximum amount of free space. In an index that will have no inserts, you might want no free space.

And again, as in a table space, free pages in the index are useful when the table receives many rows by inserts from another table, or when an application wants to insert many rows into a single page because of the same or similar clustering key. Generally, it is better to have free space in the same page than in another page, even one that's nearby. So, use FREEPAGE when it is impractical to use PCTFREE; otherwise accept its default value of 0.

In a partitioned index, you might give different values of PCTFREE or FREEPAGE for separate partitions. Give the partition number in the PART clause.

CLUSTER Clause

If you have data that needs to be viewed as a group or in sequence, create an index using the CLUSTER clause. Every partitioned table space must have one and only one clustering index.

PART: This is an integer that specifies the highest value of an index key in one partition of a partitioned index. If you use CLUSTER, and the table is contained in a partitioned table space, you must use exactly one PART clause for each partition defined with the NUMPARTS clause on the CREATE TABLESPACE statement.

VALUES: This is a constant that you must specify at least once for each PART clause. You can use as many values as there are columns in the key.

SUBPAGES Clause

Use this to specify the number of subpages for each physical page. Use 1, 2, 4, 8, or 16; the default is 4. Type 2 indexes do not support SUBPAGES.

BUFFERPOOL, CLOSE, and DSETPASS Clauses

The considerations for using BUFFERPOOL, CLOSE, and DSETPASS clauses with CREATE INDEX are essentially the same as for using them with CREATE TABLESPACE, which is described in “CREATE TABLESPACE Statement” on page 2-87. For indexes, however, you cannot use any 32KB buffer pool.

DEFER Clause

Use DEFER YES when you are creating a nonunique index on a populated table, especially if that table is large. DEFER YES allows you to build the index later, using RECOVER INDEX. When you defer the build for DB2-managed data sets, DB2 creates the VSAM data set and adds the definition for the index to the catalog, but does not build the index. For user-managed data sets, the data set must already exist before DB2 adds the definition to the catalog.

The index is placed in RECOVER PENDING status, and SQL statements that cause DB2 to access this index fail until the index is recovered with the RECOVER INDEX utility. Building an index this way on a populated table is more efficient than building it at CREATE time. The next two steps show how to do this:

- Run RECOVER INDEX, and specify the new index as the index to be recovered.

```
RECOVER INDEX (DSN8510.NEWINDEX)
```

- Run RUNSTATS to provide accurate statistical information. Bind plans and packages that use the new index. In creating an index on a large partitioned table space, the CREATE INDEX statement has other additional considerations. See “Creating a Partitioned Index on a Large Partitioned Table Space” on page 2-103 for more information about how to do this.

REORG TABLESPACE also builds the index. However, you must reorganize the entire table space and cannot use the new index to unload the data. You would probably use REORG only if you were planning on doing a reorganization anyway.

The default for the DEFER clause is NO.

We recommend that you do not use DEFER YES when creating a unique index on a populated table. With DEFER YES, the table itself is not accessed when the CREATE INDEX statement executes; thus, any duplicate values do not cause the statement to fail. However, when you attempt to build the index with RECOVER INDEX or REORG TABLESPACE, the job fails with message DSNU340I. If this happens, you must either drop and re-create the index to be nonunique, or you must run the REPAIR utility to remove the duplicate rows.

PIECESIZE Clause

Use the PIECESIZE clause to set the maximum size of a data set in a nonpartitioned index.

The default for PIECESIZE is 2 GB for a nonpartitioned index on a table space that is not defined as LARGE, or 4 GB for a nonpartitioned index on a table space defined as LARGE.

Using indexes that consist of several smaller data sets, rather than one large data set, can reduce I/O contention on the devices that contain the indexes, which can improve performance. For more information on PIECESIZE and performance, see “Spread Data Sets of Nonpartitioning Indexes” on page 5-42.

USING Clause

When you use CREATE INDEX, specify the USING clause. If you use a storage group, consider specifying the primary and the secondary space allocation quantities (PRIQTY and SECQTY). You specify the allocation as a number of kilobytes; DB2 specifies the value to VSAM as a number of pages. For each clause, the default value is three pages. For more information about how space allocation can affect the performance of mass inserts, see “Ensure Allocation in Cylinders” on page 5-40.

The primary quantity is particularly important; so try to specify a value that avoids secondary allocation without wasting space. DB2 can extend a nonpartitioned index space to more than one data set, but only if the value of PRIQTY + 118 × SECQTY is at least 2 GB.

For nonpartitioning indexes on partitioned table spaces defined as LARGE, the value of PRIQTY + 118 × SECQTY must be at least the user-specified value of

PIECESIZE, or the default value of 4GB. If DB2 reaches the maximum number of extents before reaching the PIECESIZE or 4GB limit, the extension fails.

You might possibly reach 123 extents before reaching the limit, but 119 is more likely.

The default value of SECQTY is 10 percent of the specified PRIQTY, or three times the page size, whichever is larger.

If you partition the table space, the clustering index is also partitioned.

You can specify space for the entire index by using the USING clause, or, if the index is partitioned, you can specify 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.

Creating a Partitioned Index on a Large Partitioned Table Space

The example in Figure 48 creates an index on a large partitioned table space. Our example assumes a table STOCK was previously created. A type 2 index was created by the statement. All indexes defined on a table in a LARGE partitioned table space must be type 2. If you do not specify the TYPE option, the index is TYPE 2.

The example also specifies high key values for the composite key (ITEMNO, STORENO) for the partitions using the CLUSTER clause. When you specify the VALUES clause, the constant you specify for the last partition is enforced for a large partitioned table space. If the columns of the index are ascending, the value specified for the last partition is the highest value that can be placed in the table. If the columns of the index index are descending, the value specified for the last partition is the lowest value that can be placed in the table. A key value greater than (or less than if the index is descending) the value specified for the last partition is considered to be out of range. If you want a different storage group for each partition, you can use the example in Figure 48 with the USING STOGROUP clause.

```
CREATE TYPE 2 INDEX SALIND
ON STOCK
 (ITEMNO ASC, STORENO ASC)
CLUSTER
  (PART 1 VALUES ('000999','050') USING STOGROUP MYSG1,
   PART 2 VALUES ('001999','075') USING STOGROUP MYSG2,
   PART 3 VALUES ('002999','100') USING STOGROUP MYSG3,
   PART 4 VALUES ('004999','100') USING STOGROUP MYSG4,
   PART 5 VALUES ('005999','125') USING STOGROUP MYSG5,
                .
                .
                .
   PART 66 VALUES ('018999','143') USING STOGROUP MYSG66,
   PART 67 VALUES ('020999','078') USING STOGROUP MYSG67,
   PART 68 VALUES ('022000','150') USING STOGROUP MYSG68)
                .
                .
BUFFERPOOL BP1
CLOSE YES;
```

Figure 48. Example of Creating an Index on a Large Partitioned Table Space

Creating an Index on a Large Table

Although you should try to create indexes for large tables before loading the table, there might be times when you have to create an index on a populated table. The DEFER option of CREATE INDEX can make this process more efficient by allowing you to defer the build and thus use the quicker RECOVER INDEX to build the index.

Create a new index on the table space, as shown in Figure 49.

```
CREATE INDEX DSN8510.NEWINDEX
  ON DSN8510.EMP
     (LASTNAME ASC)
  USING STOGROUP DSN8G510
     PRIQTY 3072
     SECQTY 1024
     ERASE NO
  BUFFERPOOL BP0
  CLOSE YES
  DEFER YES;
```

Figure 49. Example of Creating an Index on a Large Table

Because this is an index on a populated table, DEFER YES puts the index into recovery pending status. Any SQL statements that cause this index to be accessed will fail.

Implementing Referential Constraints

Implementing a referential constraint can be complex. For example, objects with referential constraints must sometimes be defined in a certain order. A foreign key cannot be defined unless the corresponding primary key already exists and has a unique index defined on it.

More information about implementing referential constraints in your design, such as creating table spaces and table, can be found in “Chapter 2-9. Implementing Your Design” on page 2-79.

Order of Operations in Building a Referential Structure

Without referential constraints, there is little to say about the order of creating a set of table spaces, tables, and indexes such as the one that makes up the sample application. A table space must be created before any tables it is to contain, or the table space can be created implicitly when its table is created. Tables must be created before their indexes. After they are created, the tables can be loaded in any order.

With referential constraints, parent tables must be created, with their primary and unique keys and corresponding indexes, before matching foreign keys can be defined on dependent tables.

Creating the Tables

Create table spaces in any order before you perform the following steps.

1. Start with the cycle containing the department and the employee tables—both have dependents, and neither has any other parent. Take the following steps:
   a. Create the department table and define its primary key (DEPTNO).
   b. Create the primary index for the department table.
   c. Create the employee table, defining its primary and foreign keys at the same time.
   d. Alter the department table twice, to add the definitions of its two foreign keys.
2. Create the activity table with its primary key and then define its index. This step is independent of most of the others; you could do it first, or at any time before defining the project activity table.
3. Create the project table with its primary key and the foreign keys on DEPTNO and RESPEMP, but not the foreign key on MAJPROJ. The key on MAJPROJ makes the table self-referencing, so it must be defined in a later step. At any later time, alter the table to define the last foreign key.
4. Create the project activity table with its primary and two foreign keys.
5. Create the employee to project activity table with its keys.

Loading the Tables

When loading a table, you can allow LOAD to enforce referential constraints or you can suspend enforcement by using the ENFORCE(NO) option. Because the sample tables contain a cycle that spans more than one table space, the referential constraints should be enforced *after* the tables are loaded.

1. Load all of the tables, specifying ENFORCE(NO).
2. Create an exception table for each dependent table.
3. Use ALTER TABLE to add a RID and timestamp column to each exception table.
4. Run CHECK DATA against all the table spaces, specifying DELETE YES.
5. Correct any erroneous rows in the exception tables.
6. Insert the corrected rows into the original tables using INSERT.

Implementing Your Views

In designing your database, you may find it necessary to give users access to only certain pieces of data. This can be done with the design and use of *views*.

Use the CREATE VIEW statement to define a view and give the view a name. Unless you specifically list different column names after the view name, the column names of the view are the same as those of the underlying table. (Table 15 on page 2-107 shows an example of this.) When creating different column names for your view, remember the naming conventions you established when designing the relational database.

As the following examples illustrate, the information in the view is described by a SELECT statement. The SELECT statement can name other views as well as tables, and can use the WHERE, GROUP BY, and HAVING clauses. It cannot use the ORDER BY clause or name a host variable.

You cannot create a view on a table from another subsystem. However, with the proper authority, you can access a view from another subsystem.

Creating a View on a Single Table

The example below illustrates creating a view on a single table, the department table. Of the four columns in the table, only three are required for the view: DEPTNO, DEPTNAME, and MGRNO. The order of the columns in the SELECT clause is the order in which they appear in the view.

```
CREATE VIEW VDEPT3 AS
  SELECT DEPTNO,DEPTNAME,MGRNO
  FROM DSN8510.DEPT;
```

In this example, because no column list follows the view name, VDEPT3, the columns of the view have the same names as those of the table on which it is based (DEPTNO, DEPTNAME, MGRNO). Table 14 is the result of executing this SQL statement:

```
SELECT * FROM VDEPT3;
```

Table 14. A View of a Table

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

Creating a View Combining Information from Several Tables

DB2 provides two types of joins, an *outer join* and an *inner join*. An outer join includes rows where the values in the join columns do not match, as well as rows where the values match. An inner join includes only rows where matching values in the join columns are returned. For more information about outer join, see Section 2 of *Application Programming and SQL Guide*. An inner join is shown in the next example, which includes the manager's name (from table DSN8510.EMP) along with information from DSN8510.DEPT. The WHERE clause shown limits the view to just those columns where the MGRNO in the DSN8510.DEPT table matches the EMPNO in the DSN8510.EMP table.

```
CREATE VIEW SMITH.VDEPTM  AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
  FROM DSN8510.DEPT, DSN8510.EMP
  WHERE DSN8510.EMP.EMPNO = DSN8510.DEPT.MGRNO;
```

Table 15 on page 2-107 shows the result of executing the CREATE VIEW statement:

Table 15. Inner Join View from Two Tables

DEPTNO	MGRNO	LASTNAME	ADMRDEPT
A00	000010	HAAS	A00
B01	000020	THOMPSON	A00
C01	000030	KWAN	A00
E01	000050	GEYER	A00
D11	000060	STERN	D01
D21	000070	PULASKI	D01
E11	000090	HENDERSON	E01
E21	000100	SPENSER	E01

Now, suppose you want to create the same view, but including only those departments that report administratively to Department A00. Suppose also that you want a different set of column names. The CREATE statement is shown below.

```
CREATE VIEW SMITH.VDEPTMA00
  (DEPT, MGR, NAME, REPORTTO)
  AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
  FROM DSN8510.EMP, DSN8510.DEPT
  WHERE DSN8510.EMP.EMPNO = DSN8510.DEPT.MGRNO
  AND ADMRDEPT = 'A00';
```

The result of SELECT * FROM SMITH.VDEPTMA00 is shown in Table 16.

Table 16. View Created with New Column Names

DEPT	MGR	NAME	REPORTTO
A00	000010	HAAS	A00
B01	000020	THOMPSON	A00
C01	000030	KWAN	A00
E01	000050	GEYER	A00

Inserting and Updating through Views

You can name a view in an INSERT or UPDATE statement; the insert or update is made to the base table. (You cannot insert into or update a view that is based on more than one table.)

Unless the view definition uses the WITH CHECK OPTION clause, no check is made to see that the insert or update conforms to the definition of the view. The following example illustrates some probably undesirable results of omitting that check.

Suppose the view V1 is defined as follows:

```
CREATE VIEW V1 AS
  SELECT * FROM DSN8510.EMP
    WHERE WORKDEPT LIKE 'D%' ;
```

A user with the SELECT privilege on V1 can see the information from the employee table for employees in departments whose IDs begin with 'D': D01, D11, or D21.

Somewhat surprisingly, a user with the INSERT privilege on V1 can insert a new row with any value for WORKDEPT, whether it begins with a 'D' or not: a user with both SELECT and INSERT privileges can insert a row for department E01, perhaps erroneously, and not be able to select the row just inserted.

WITH CHECK OPTION for Views

To avoid the situation in which a value that does not match the view definition is inserted into the base table, modify the definition of view V1 to include WITH CHECK OPTION. The definition of view V1 then becomes:

```
CREATE VIEW V1 AS
  SELECT * FROM DSN8510.EMP
    WHERE WORKDEPT LIKE 'D%'
  WITH CHECK OPTION;
```

With the new definition, any insert or update made to V1 must satisfy the predicate contained in the WHERE clause: WORKDEPT LIKE 'D%'. The check can be valuable, but it also carries a processing cost; each potential insert or update must be checked against the view definition.

Views of Views, With and Without Checking

Now suppose there is a second view defined on the first view. For example:

```
CREATE VIEW V2 AS
  SELECT * FROM V1
    WHERE WORKDEPT IN ('B01', 'C01', 'D01');
```

A user who selects from V2 retrieves data only for employees in department D01, the only department number permitted in both views. But a user who inserts into V2 might be able to enter rows for one department, three departments, or all departments, depending on whether one, both, or neither of V1 and V2 were defined using WITH CHECK OPTION. The following table shows the possibilities:

Table 17. Predicates Checked When Inserting Into or Updating a View of a View

View 1	View 2	Predicates Checked
CHECK	CHECK	View 1 and view 2
CHECK	-	View 1
-	CHECK	View 2
-	-	None

In the table, the entry CHECK indicates that the view uses checking, specified by the WITH CHECK OPTION clause. For example, if View 1 and View 2 are the views V1 and V2 defined above, and if both use checking, an update of a department ID through V2 must satisfy the predicates of both V1 and V2. Hence, it can only be D01. If only V1 uses checking, the update value must be D01, D11, or D12; if only V2 uses checking, the update value must be B01, C01, or D01. And if neither view uses checking, the department ID can be updated, through V2, to any value the column can hold.

Using LOCAL or CASCADED CHECK OPTION to Control Updates on Views of Views

Views and their underlying views might all have search conditions. You can use the value LOCAL or CASCADED in WITH CHECK OPTION to determine whether DB2 applies search conditions on underlying views when you perform an insert or update operation on a view. If you defined the view:

WITH CASCADED CHECK OPTION
All search conditions of underlying views are checked unconditionally, that is, whether or not those underlying views are defined with a check option.

WITH LOCAL CHECK OPTION
Search conditions of underlying views are checked conditionally, that is, only if they are defined with a check option or if they are underlying views of a view defined with a check option.

The difference between CASCADE and LOCAL is shown best by example. Consider the following updatable views, where x and y represent either LOCAL or CASCADE:

V3 is defined on Table T0.
V4 is defined on V3 WITH x CHECK OPTION.
V5 is defined on V4.
V6 is defined on V5 WITH y CHECK OPTION.
V7 is defined on V6.

Table 18 shows the views in which search conditions are checked during an INSERT or UPDATE operation:

Table 18. Views in Which Search Conditions are Checked during INSERT and UPDATE Operations

View used in INSERT or UPDATE Operation	x = LOCAL y = LOCAL	x = CASCADED y = CASCADED	x = LOCAL y = CASCADED	x = CASCADED y = LOCAL
V3	none	none	none	none
V4	V4	V4, V3	V4	V4, V3
V5	V4	V4, V3	V4	V4, V3
V6	V6, V4	V6, V5, V4, V3	V6, V5, V4, V3	V6, V4, V3
V7	V6, V4	V6, V5, V4, V3	V6, V5, V4, V3	V6, V4, V3

Creating Schemas

One way to organize your CREATE TABLE, CREATE VIEW and GRANT (table privileges) statements is to put them in a *schema definition*. A schema definition is a set of CREATE TABLE, CREATE VIEW and GRANT statements that are preceded by the words CREATE SCHEMA.

CREATE SCHEMA is not an SQL statement, and 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 2-110. 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.

The order of statements within a schema definition is important.

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 as long as the object table is created within the same CREATE SCHEMA. By relaxed we mean that the requirement that all referenced objects have been previously created is not checked until all of the statements have been processed. For example, this means that 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 50 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 50. 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 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 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 come before the CREATE SCHEMA statement; the end of input ends the schema definition. SQL comments can be used within and between SQL statements as well.

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. You need to fix the statement in error and resubmit the entire schema definition.

Chapter 2-10. Loading Data into DB2 Tables

This chapter provides an overview of how to load data into DB2 tables. There are several ways to fill DB2 tables with data, but you will probably load most of your tables 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 Section 2 of *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, then the table space being loaded, its index spaces, and even other table spaces can be left in a restricted status.

  Plan to make necessary corrections and remove restrictions after any such LOAD job. For instructions, see 2-114.

- An SQL INSERT statement in an application program. See "Loading Data Using the SQL INSERT Statement" on page 2-115 and *SQL Reference*. The method allows you to develop an application 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 2-115 and *SQL Reference*.

To reformat data from IMS DL/I databases and VSAM and SAM loading for the LOAD utility, use the DataPropagator Relational licensed program. See "Loading Data from DL/I" on page 2-116.

For general guidance about running DB2 utility jobs, see *Utility Guide and Reference*. For information about DataPropagator Relational, see *DataPropagator Relational User's Guide*.

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 perform LOAD.

LOAD loads records into the tables and builds or extends any indexes defined on them. If the table space already contains data, you can choose whether you want to add the new data to the existing data or replace the existing data.

Data input with LOAD can be in either the ASCII or EBCDIC character encoding schemes.

For nonpartitioned table spaces, or if there are nonpartitioned indexes defined on a table in 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 a table is loaded, the default value of a column not loaded is the value specified in the DEFAULT clause of the ALTER TABLE or CREATE TABLE statements, if there is one.

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 turn off the restriction, by making a full image copy using SHRLEVEL REFERENCE. (If you end the copy job before it is finished, the table space is still in copy pending status.)

| When REORG or LOAD REPLACE is used and the COPYDDN keyword is specified, 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 not valid unless the entire table space or partition is being replaced. If an inline copy is requested by specifying 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 “recovery pending” status if you end the job (using -TERM UTILITY) before it completes the INDEXVAL phase. It places the table space itself in “recovery pending” if you end the job before it completes the RELOAD phase.

LOAD places a table space in the “check pending” status if its referential or check integrity is in doubt. The intent of the restriction is to encourage the use of the

CHECK DATA utility. That utility locates invalid data and, optionally, removes it. If it removes the invalid data, the data remaining satisfies all referential and table check constraints, and the check pending restriction is lifted.

Loading Data Using the SQL INSERT Statement

The information under this heading, up to “Loading Data from DL/I” on page 2-116 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

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 created 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 DSN8D51A.DSN8S51D;
```

To add a row to table TEMPDEPT, you might 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 DSN8510.DEPT
  WHERE ADMRDEPT='D01';
```

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

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.

For the full syntax of the statement, see *SQL Reference*.

Loading Data from DL/I

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

Chapter 2-11. Using the Catalog in Database Design

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

Retrieving information from the catalog, using SQL statements, can be helpful in designing your relational database. Appendix D of *SQL Reference* lists all the DB2 catalog tables and the information stored in them.

The information in the catalog is vital to normal DB2 operation. As the examples in this chapter show, you can *retrieve* catalog information, but *changing* it could have serious consequences. So you cannot execute INSERT or DELETE statements that affect the catalog, and there is only a limited list of columns you can update. Exceptions to these restrictions are the SYSIBM.SYSSTRINGS, SYSIBM.SYSPROCEDURES, SYSIBM.SYSCOLDIST, and SYSIBM.SYSCOLDISTSTATS catalog tables, into which you can insert rows and proceed to update and delete rows. See “Chapter 5-9. Maintaining Statistics in the Catalog” on page 5-243 for information about updating catalog columns.

To execute the following examples, you need at least the SELECT privilege on the appropriate catalog tables. Be careful with your own examples; querying the DB2 catalog can result in a long table space scan.

Retrieving Catalog Information about DB2 Storage Groups

SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES contain information about DB2 storage groups and the volumes in those storage groups. The following query shows what volumes are in a DB2 storage group, how much space is used, and when that space was last calculated.

```
SELECT SGNAME,VOLID,SPACE,SPCDATE
  FROM SYSIBM.SYSVOLUMES,SYSIBM.SYSSTOGROUP
  WHERE SGNAME=NAME
  ORDER BY SGNAME;
```

Retrieving Catalog Information about a Table

SYSIBM.SYSTABLES contains a row for every table, view, and alias in your DB2 system. For each, the row tells you whether the object is a table, a view, or an alias, its name, who created it, what database it belongs to, what table space it belongs to, and other information. SYSTABLES also has a REMARKS column in which you can store your own information about the table in question. See “Adding and Retrieving Comments” on page 2-121 for more information about how to do this.

This statement displays all the information for the project activity sample table:

```
SELECT *
  FROM SYSIBM.SYSTABLES
  WHERE NAME = 'PROJACT'
  AND CREATOR = 'DSN8510';
```

Retrieving Catalog Information about Aliases

SYSIBM.SYSTABLES describes the aliases you create. It has three columns used only for aliases:

- LOCATION contains your subsystem's location name for the remote system, if the object on which the alias is defined resides at a remote subsystem.
- TBCREATOR contains the owner of the table or view.
- TBNAME contains the name of the table or the view.

The NAME and the CREATOR columns of SYSTABLES contain the name and owner of the alias, and three other columns contain the following alias information:

- TYPE will be A.
- DBNAME will be DSNDB06.
- TSNAME will be SYSDBAUT.

If similar tables at different locations have names with the same second and third parts, you can retrieve the aliases for them with a query like this one:

```
SELECT LOCATION, CREATOR, NAME
  FROM SYSIBM.SYSTABLES
    WHERE TBCREATOR='DSN8510' AND TBNAME='EMP'
      AND TYPE='A';
```

Retrieving Catalog Information about Columns

SYSIBM.SYSCOLUMNS has one row for each column of every table and view. Query it, for example, if you cannot remember the column names of a table or view.

This statement retrieves information about columns in the sample Department table:

```
SELECT NAME, TBNAME, COLTYPE, LENGTH, NULLS, DEFAULT
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBNAME='DEPT'
  AND TBCREATOR = 'DSN8510';
```

The result is shown below; it tells for each column:

- The column name
- The name of the table that contains it
- Its data type
- Its length attribute
- Whether or not it allows nulls
- Whether or not it allows default values.

```
NAME       TBNAME  COLTYPE          LENGTH  NULLS  DEFAULT
DEPTNO     DEPT    CHAR                  3    N       N
DEPTNAME   DEPT    VARCHAR              36    N       N
MGRNO      DEPT    CHAR                  6    Y       N
ADMRDEPT   DEPT    CHAR                  3    N       N
```

Retrieving Catalog Information about Indexes

SYSIBM.SYSINDEXES contains a row for every index, including indexes on catalog tables. This example retrieves information about the index XEMPL2.

```
SELECT *
  FROM SYSIBM.SYSINDEXES
  WHERE NAME = 'XEMPL2'
  AND CREATOR = 'DSN8510';
```

The previous example displays a single row of information about a particular index. But a table can have more than one index. To display information about all the indexes of a table, enter a statement like this one:

```
SELECT *
  FROM SYSIBM.SYSINDEXES
  WHERE TBNAME = 'EMP'
  AND TBCREATOR = 'DSN8510';
```

Retrieving Catalog Information about Views

For every view you create, DB2 stores descriptive information in several catalog tables. The following is what happens in the catalog following the execution of CREATE VIEW:

- A row is inserted into SYSIBM.SYSTABLES.
- A row is inserted into SYSIBM.SYSTABAUTH to record the owner's privileges on the view.
- For each column of the view, a row is inserted into SYSIBM.SYSCOLUMNS.
- One or more rows are inserted into the SYSIBM.SYSVIEWS table to record the text of the CREATE VIEW statement.
- For each table or view on which the view is dependent, a row is inserted into SYSIBM.SYSVIEWDEP to record the dependency.
- A row is inserted into SYSIBM.SYSVTREE, and possibly into SYSIBM.SYSVLTREE, to record the parse tree of the view (an internal representation of its logic).

Users might want a view of one or more of those tables, containing information about their own tables and views.

Retrieving Catalog Information about Authorizations

SYSIBM.SYSTABAUTH contains information about the privileges held by authorization IDs over tables and views. Query it to learn who can access your data. The following query retrieves the names of all users who have been granted access to the DSN8510.DEPT table.

```
SELECT GRANTEE
  FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'DEPT'
    AND GRANTEETYPE <> 'P'
    AND TCREATOR = 'DSN8510';
```

GRANTEE is the name of the column that contains authorization IDs for users of tables. TTNAME and TCREATOR specify the DSN8510.DEPT table. The clause GRANTEETYPE <> 'P' ensures that you retrieve the names only of users (not application plans or packages) that have authority to access the table.

Retrieving Catalog Information about Primary Keys

SYSIBM.SYSCOLUMNS identifies columns of a primary key in column KEYSEQ; a nonzero value indicates the place of a column in the primary key. To retrieve the creator, database and names of the columns in the primary key of the sample project activity table using SQL statements, execute:

```
SELECT TBCREATOR, TBNAME, NAME, KEYSEQ
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBCREATOR = 'DSN8510'
  AND TBNAME = 'PROJACT'
  AND KEYSEQ > 0
    ORDER BY KEYSEQ;
```

SYSIBM.SYSINDEXES identifies the primary index of a table by the value 'P' in column UNIQUERULE. To find the name, creator, database, and index space of the primary index on the project activity table, execute:

```
SELECT TBCREATOR, TBNAME, NAME, CREATOR, DBNAME, INDEXSPACE
  FROM SYSIBM.SYSINDEXES
  WHERE TBCREATOR = 'DSN8510'
  AND TBNAME = 'PROJACT'
  AND UNIQUERULE = 'P';
```

Retrieving Catalog Information about Foreign Keys

SYSIBM.SYSRELS contains information about referential constraints, and each constraint is uniquely identified by the creator and name of the dependent table and the constraint name (RELNAME). SYSIBM.SYSFOREIGNKEYS contains information about the columns of the foreign key that defines the constraint. To retrieve the constraint name, column names, and parent table names for every relationship in which the project table is a dependent, execute:

```
SELECT A.CREATOR, A.TBNAME, A.RELNAME, B.COLNAME, B.COLSEQ,
       A.REFTBCREATOR, A.REFTBNAME
  FROM SYSIBM.SYSRELS A, SYSIBM.SYSFOREIGNKEYS B
  WHERE A.CREATOR = 'DSN8510'
  AND B.CREATOR = 'DSN8510'
  AND A.TBNAME = 'PROJ'
  AND B.TBNAME = 'PROJ'
  AND A.RELNAME = B.RELNAME
    ORDER BY A.RELNAME, B.COLSEQ;
```

You can use the same tables to find information about the foreign keys of tables to which the project table is a parent, as follows:

```
SELECT A.RELNAME, A.CREATOR, A.TBNAME, B.COLNAME, B.COLNO
  FROM SYSIBM.SYSRELS A, SYSIBM.SYSFOREIGNKEYS B
  WHERE A.REFTBCREATOR = 'DSN8510'
  AND A.REFTBNAME = 'PROJ'
  AND A.RELNAME = B.RELNAME
    ORDER BY A.RELNAME, B.COLNO;
```

Retrieving Catalog Information about Check Pending

SYSIBM.SYSTABLESPACE indicates that a table space is in check pending status by a value in column STATUS: 'P' if the entire table space has that status, 'S' if the status has a scope of less than the entire space. To list all table spaces whose use is restricted for *any* reason, give this command:

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

To retrieve the names of table spaces in check pending status only, with the names of the tables they contain, execute:

```
SELECT A.DBNAME, A.NAME, B.CREATOR, B.NAME
  FROM SYSIBM.SYSTABLESPACE A, SYSIBM.SYSTABLES B
  WHERE A.DBNAME = B.DBNAME
  AND A.NAME = B.TSNAME
  AND (A.STATUS = 'P' OR A.STATUS = 'S')
    ORDER BY 1, 2, 3, 4;
```

Retrieving Catalog Information about Table Check Constraints

The following query shows all table check constraints on all tables named SIMPDEPT and SIMPEMPL in order by column name within table owner. It shows the name, authorization ID of the creator, and text for each constraint. A constraint that uses more than one column name appears more than once in the result. Information about check constraints is stored in the DB2 catalog in:

- SYSIBM.SYSCHECKS, which contains one row for each check constraint defined on a table
- SYSIBM.SYSCHECKDEP, which contains one row for each reference to a column in a check constraint

```
CREATE TABLE SIMPDEPT
  (DEPTNO   CHAR(3) NOT NULL,
   DEPTNAME VARCHAR(12) CONSTRAINT CC1 CHECK (DEPTNAME IS NOT NULL),
   MGRNO    CHAR(6),
   MGRNAME  CHAR(6));
```

```
SELECT A.TBOWNER, A.TBNAME, B.COLNAME,
A.CHECKNAME, A.CREATOR, A.CHECKCONDITION
FROM SYSIBM.SYSCHECKS A, SYSIBM.SYSCHECKDEP B
WHERE A.TBOWNER = B.TBOWNER
  AND A.TBNAME = B.TBNAME
  AND B.TBNAME = 'SIMPDEPT'
  AND A.CHECKNAME = B.CHECKNAME
  ORDER BY TBOWNER, TBNAME, COLNAME;
```

Adding and Retrieving Comments

After you create a table, a view, or an alias, you can provide explanatory information about it for future reference—information such as the purpose of the table, who uses it, and anything unusual about it. Not only can you store a comment about the table or the view as a whole, you can also include one for *each column*. A comment must not exceed 254 bytes.

A comment is especially useful if your names do not clearly indicate the contents of columns or tables. In that case, use a comment to describe the specific contents of the column or table.

Below are two examples of COMMENT ON:

```
COMMENT ON TABLE DSN8510.EMP IS
  'Employee table. Each row in this table represents one
   employee of the company.';

COMMENT ON COLUMN DSN8510.PROJ.PRSTDATE IS
  'Estimated project start date. The format is DATE.';
```

After executing a COMMENT ON statement, your comments are stored in the REMARKS column of SYSIBM.SYSTABLES or SYSIBM.SYSCOLUMNS. (Any comment already present in the row is replaced by the new one.) The next two examples retrieve the comments added by the previous COMMENT ON statements.

```
SELECT REMARKS
  FROM SYSIBM.SYSTABLES
  WHERE NAME = 'EMP'
  AND CREATOR = 'DSN8510';

SELECT REMARKS
  FROM SYSIBM.SYSCOLUMNS
  WHERE NAME = 'PRSTDATE' AND TBNAME = 'PROJ'
  AND TBCREATOR = 'DSN8510';
```

Verifying the Accuracy of the Database Definition

The catalog can also be used to verify the accuracy of your database definition process. After you have created the objects in your database, display selected information from the catalog to check that there were no errors in your CREATE statements. By displaying the catalog tables, you can verify that you have the correct tables in each table space, the table space associated with the correct storage group, and so on.

Chapter 2-12. Altering Your Database Design

The information under this heading, up to “Changing the High-Level Qualifier for DB2 Data Sets” on page 2-139 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

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, in:

  “Altering DB2 Storage Groups” on page 2-124
  “Altering DB2 Databases” on page 2-125
  “Altering Table Spaces” on page 2-125
  “Altering Tables” on page 2-128
  “Altering Indexes” on page 2-137
  “Altering Views” on page 2-138

- Data set passwords, in “Changing Data Set Passwords” on page 2-139
- Data set high-level qualifier, in “Changing the High-Level Qualifier for DB2 Data Sets” on page 2-139

  This task is actually a series of subtasks and includes procedures for changing the VCAT name for DB2-managed and user-managed data sets, system data sets, and for moving data to a different storage group.
- The location of DB2 data, in “Moving DB2 Data” on page 2-147.

You can alter the definition of a DB2 object by either:

- 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 with ALTER. Application and object registration tables can restrict the use of ALTER. See “Chapter 3-3. Controlling Access Through a Closed Application” on page 3-49 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 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 invalid 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 Section 4 of *Application Programming and SQL Guide*. For more information about dropping a table, see “Implications of Dropping a Table” on page 2-134.

Altering DB2 Storage Groups

You can use the ALTER STOGROUP statements to add or remove volumes from a storage group. If you want to migrate to another device type or change the integrated catalog facility catalog name, you need to move the data. See “Moving DB2 Data” on page 2-147 for more information.

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 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 DSN8G510, the DB2 data on that volume remains intact. However, when a new table space is defined using DSN8G510, 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 DSN8G510:

   ```
   SELECT TSNAME, DBNAME
     FROM SYSIBM.SYSTABLEPART
     WHERE STORNAME ='DSN8G510' 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 have to be extended. You can do this by stopping the database.
4. Use the ALTER STOGROUP statement to remove the volume associated with the old storage group and to add the new volume.

   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.

   ```
   ALTER STOGROUP DSN8G510
     REMOVE VOLUMES (VOL1)
     ADD VOLUMES (VOL2);
   ```

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 TABLESPACE *dbname.tsname*.
6. Start the database.

Altering DB2 Databases

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

- STOGROUP. Allows you to change the name of the default storage group to support DASD space requirements for table spaces and indexes within the database. The new default DB2 storage group is only used for new table spaces and indexes; existing definitions are not changed.
- BUFFERPOOL. Allows you to 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 are not changed.
- ROSHARE. Allows you to convert an existing database from not shared to shared, or vice versa. For more information about converting to or from a shared database, see “Altering” on page X-169.

Altering Table Spaces

This section discusses the following topics:

- The attributes of a table space that you can change with ALTER TABLESPACE, in “Using the ALTER TABLESPACE Statement”
- How to change the space allocation of a table space for which the data sets are user-managed, in “Changing the Space Allocation for User-Managed Data Sets” on page 2-127
- How to change table space attributes when you cannot use ALTER TABLESPACE, in “Dropping, Re-creating, or Converting a Table Space” on page 2-127

Using the ALTER TABLESPACE Statement

The ALTER TABLESPACE statement allows you to change these clauses:

- BUFFERPOOL. Allows you to name the bufferpool to be associated with the table space. The page size must remain the same. The change does not take effect until the data set is closed and reopened.

  (To change the *size* of the buffer pool, you can use the ALTER BUFFERPOOL command, as described in Chapter 2 of *Command Reference* .)
- LOCKSIZE. Allows you to specify the locking level for the table space. You can use the options PAGE, TABLESPACE, or ANY, and if there are no type 1 indexes on the table space, you can use ROW. For a segmented table space, you can use TABLE.

  If LOCKSIZE TABLESPACE is specified, LOCKMAX must be omitted or its operand must be 0.
- LOCKMAX. Allows you to specify the number of locks for the table space.
- CLOSE. The CLOSE clause tells the priority DB2 should use when determining which open data sets to close.
- DSETPASS. Allows you to specify the password that is passed to VSAM when the data sets of the table space are used by DB2. You must also use VSAM access method services to change the data set passwords. Be cautious about

changing passwords: be sure the table space is stopped and no activity is pending on it.

- PART. Allows you to identify a partition of the table space.
- FREEPAGE. Allows you to specify how often to leave a page of free space when the table space is loaded or reorganized.
- PCTFREE. Allows you to specify what percentage of each page to leave as free space when the table space is loaded or reorganized.
- USING. You can change from one group of data sets to another, or between user-managed and DB2-managed data sets, by changing the value of VCAT (the integrated catalog facility catalog name) or STOGROUP. The change has no immediate effect on the existing data sets; the new data sets are used when you RECOVER, REORG, or LOAD REPLACE the table space, or when DB2 extends to a new data set. For suggestions on moving the data to a new device, see "Moving DB2 Data" on page 2-147.
- PRIQTY. Allows you to specify the primary space allocation for a data set of the table space or partition when the data sets for the table space are DB2-managed.
- SECQTY. Allows you to specify the secondary space allocation for a data set of the table space or partition, when the data sets for the table space are DB2-managed.
- ERASE. Allows you to specify whether the contents of a data set for the table space or partition are erased when the table space is dropped.

  You can change the options for PRIQTY, SECQTY, and ERASE in either the same or a new storage group. You can change the storage attribute for a partition of the table space by using the PART clause.
- COMPRESS. Allows you to specify whether the contents of a data set for the table space or partition are compressed. For more information about compression, see "Compressing Data in a Table Space or Partition" on page 2-63.
  - To compress data in an existing table space, use ALTER TABLESPACE with COMPRESS YES, then use REORG TABLESPACE (or LOAD with REPLACE or RESUME NO) to build the compression dictionary and compress the data.
  - To decompress data, use ALTER TABLESPACE with COMPRESS NO, then use REORG TABLESPACE (or LOAD). This erases the compression dictionary and decompresses the data in the table space or partition.
- GBPCACHE. Allows you to specify what pages of the table space or partition are written to the group buffer pool in a data sharing environment. In a non-data-sharing environment, you can specify this option, but it is ignored.
- MAXROWS. Allows you to specify the number of rows that DB2 will consider placing on each page, where the *integer* can range from 1 to 255. The change takes effect immediately for new rows added. It is highly recommended to reorganize the table space after altering MAXROWS.
- LOCKPART. Allows you to indicate whether selective partition locking is to be used when locking a partitioned table space.

Changing the Space Allocation for User-Managed Data Sets

If the table space is supported by data sets that are user-managed, 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, then the table space must be stopped with the STOP DATABASE command with 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 to do the reload.

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 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 using 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. For each table (for example, TA1, TA2, TA3, ...) in the table space (TS1), if you do not have the original CREATE TABLE statement and all authorization statements for the table, 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
   ```
3. If necessary, unload the data, using a statement such as:

   ```
   REORG TABLESPACE DSN8D410.TSPACE LOG NO SORTDATA UNLOAD ONLY
   ```

   Or, you can insert the data from your old tables to the new tables executing a statement like:

   ```
   INSERT INTO TB1
     SELECT * FROM TA1;
   ```
4. Drop the table space, 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 use privileges. You can also create a large partitioned table space. You could use the following statements:

```
CREATE LARGE TABLESPACE TSPACE
   IN DSN8D51A
   USING STOGROUP DSN8G510
     PRIQTY 4000
     SECQTY 130
     ERASE NO
 NUMPARTS 95
  (PART 45 USING STOGROUP DSN8G510
     PRIQTY 4000
     SECQTY 130
     COMPRESS YES,
   PART 62 USING STOGROUP DSN8G510
     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 users' privileges on those tables. See “Implications of Dropping a Table” on page 2-134 for more information.
9. For each table, execute an INSERT statement of the following form:

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

10. Drop the tables TB1, TB2, TB3, ...

    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” on page 2-129.
- Change the AUDIT clause, using the options ALL, CHANGES, or NONE. For the effects of the AUDIT value, see “Chapter 3-5. Protecting Data Sets” on page 3-113.
- Add or drop a primary or a foreign key; see “Altering a Table for Referential Integrity” on page 2-130.
- Change the VALIDPROC clause; see “Altering the Assignment of a Validation Routine” on page 2-132.
- Change the DATA CAPTURE clause; see “Altering a Table for Capture of Changed Data” on page 2-133.

- Add or drop a table check constraint; see “Adding or Dropping Table Check Constraints” on page 2-132.
- Add or drop the restriction on dropping the table and the database and table space that contain the table; see *SQL Reference* .
- Use with temporary tables to add a column only; see “CREATE GLOBAL TEMPORARY TABLE Statement” on page 2-98.

In addition to the above topics, this section includes techniques for making the following changes:

- “Altering an Edit Procedure or Field Procedure” on page 2-133
- “Altering the Subtype of a String Column” on page 2-134.

For other changes, you must drop and re-create the table as described in “Altering Data Types and Attributes and Deleting Columns” on page 2-134.

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, in order 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 there will be some performance degradation immediately. To change the records to fixed-length, do the following:

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 made before the REORG you ran in step 1.
3. If you add a column that specifies PRIMARY KEY or UNIQUE, then, a unique index should be created.

Inserting values in the new column could 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 if you use 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 short string column, you can specify a field procedure for it; see “Field Procedures” on page X-57. If you do specify a field procedure, you cannot also specify NOT NULL.

The following example adds a new column to the table DSN8510.DEPT that contains a location code for the department. The column name is LOCNCODE, and its data type is CHAR (4).

```
ALTER TABLE DSN8510.DEPT
  ADD LOCNCODE CHAR (4);
```

Altering a Table for Referential Integrity

If you plan to let DB2 enforce referential integrity in a set of tables, then you must read “Chapter 2-3. Maintaining Data Integrity” on page 2-19. That section describes the requirements for referential constraints.

Adding Referential Constraints to Existing Tables

For this illustration, 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 procedure conforms to these rules:

- 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.

There is more than one sequence of operations that could build the same referential structure. 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.

   For each table except the activity table, you need to issue an ALTER TABLE statement to add its foreign keys. 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 could be disastrous. For example, if you managed to delete the row for department A00 from the department table, it 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, and correct any errors as before. If there are errors, be particularly careful not to make any row inconsistent with the department table when correcting them.
6. Again add the foreign key on MGRNO to the department table. This again leaves the table space in check pending status, so again 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.

Implications of Adding Primary and Foreign Keys

Adding a primary 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. That index, or the first such one encountered, becomes the primary index. Because of the unique index, there are no duplicate values of the key in the table, therefore checking for the validity of the data is not needed.

  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 DSN8510.DEPT
     ADD PRIMARY KEY (DEPTNO);
  ```

- 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 "Chapter 2-3. Maintaining Data Integrity" on page 2-19. 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 instance, you need to execute CHECK DATA to clear the check pending status.

Implications of Dropping Primary and Foreign Keys

Dropping a foreign key drops the corresponding referential relationship and requires the ALTER privilege on the dependent table, and either the ALTER or REFERENCES privilege on the parent table.

When you drop a primary key, DB2 drops all the referential relationships in which
the table is a parent, and requires 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 is still a unique index.

In the case of dropping a foreign key or a primary key, you should 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. It is likely that some of your application programs depend on that. The foreign key defines a referential relationship and a delete rule. Without the key, your application programs must enforce the constraints.

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 'STD', then the new check constraint is enforced immediately. If any rows violate the new check constraint, an error occurs and the description of the table is unchanged. If no rows violate the new check constraint, the check constraint is added to the description of the table.
- If 'DB2', then enforcement of the new check constraint is deferred and the table space or partition containing the table is placed in a check pending state because check integrity cannot be guaranteed.

The ALTER TABLE statement used to define a check constraint always fails if the table space or partition that contains the table is in a check pending state, 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.

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 will no longer be given control when DB2 accesses the
| table. For example:

```
ALTER TABLE DSN8510.EMP
  VALIDPROC NULL;
```

- Assign a new validation routine to the table, using the VALIDPROC *program-name* 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.) Rows that existed before the connection of a new validation routine are not validated. For example:

```
ALTER TABLE DSN8510.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 DSN8D51A.DSN8S51E
     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 DSN8510.EMP
      REPLACE
      FORMAT UNLOAD
      DISCARDDN SYSDISC
   ```

All rows are validated by the EMPLNEWE validation routine after the LOAD step has completed. Any invalid rows would have been copied 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 that can be retrieved by a program such as the log apply feature of the Remote Recovery Data Facility (RRDF) program offering, or DataPropagator Relational.

To return a table back to normal logging, use DATA CAPTURE NONE.

Altering an Edit Procedure or Field Procedure

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

1. Run REORG with UNLOAD PAUSE on the table space that contains the table whose edit or field procedure is to change.

   If you are using the same edit or field procedure for multiple columns, make sure you run REORG on all the table spaces that have tables which use the procedure.

2. Change the edit or field procedure code.
3. After the UNLOAD phase of REORG (at the pause), stop DB2.
4. Link-edit the new procedure with the same name as the old one.
5. Start DB2.
6. Restart the paused REORG.

Altering the Subtype of a String Column

If you add a column with a string data type, as described in "String Data Types" on page 2-45, you can specify its subtype in the ALTER TABLE statement. Subtypes are valid only for string columns of data types CHAR, VARCHAR, and LONG VARCHAR.

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 upon 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 Attributes and Deleting Columns

Some changes to a table cannot be made with an ALTER TABLE statement. For example, you might need to change the data type of a column to make its fields longer. An original specification of CHAR (20), for example, must become CHAR (25) or a column defined as SMALLINT now must be INTEGER. Or, a column defined with NOT NULL must now admit null values.

To make such changes, you need to drop the table, commit the drop, and re-create the table. *Be very careful about dropping a table*—in most cases, it is nearly impossible to recover a dropped table. If you do decide to drop a table, remember that such changes might cause a plan or a package to be marked invalid as described in "Dropping and Re-creating DB2 Objects" on page 2-123.

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

Implications of Dropping a Table

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

```
DROP TABLE DSN8510.PROJ;
```

The statement deletes the row in the SYSIBM.SYSTABLES catalog table that contains information about DSN8510.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.

- 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 the case of the project table, it is no longer a dependent of the department and employee tables, nor a parent of the project activity table.
- Authorization information 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 or might not be reclaimed.

  If the table space containing the table is:

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

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

  – Segmented, then DB2 reclaims the space.

  – Simple, and contains other tables as well, then you have to run the REORG utility to reclaim the space.

If a table has a partitioned 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, it is a good idea to check 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 = 'DSN8510'
  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 = 'DSN8510'
  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 = 'DSN8510'
  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.

Recreating 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;
   ```

   to copy 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 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), follow the same steps except:

- Instead of steps 2 and 3, use an application program to copy the data into a non-DB2 data set. While copying the data, convert it as needed, for example from numeric to string.
- Instead of steps 7 and 8, use an application program to copy the converted data into the new table.

Another alternative is to use the sample program DSNTIAUL to save the data in a sequential file and use the LOAD utility to repopulate the table after re-creating it.

This method is particularly appealing when you are trying to re-create a large table where DB2 DASD is constrained. When you reload the table, make sure you edit the LOAD statement to match the new column definition.

Altering a Table from EBCDIC to ASCII

If you want to convert a table from EBCDIC to ASCII, follow these steps:

1. Quiesce all activity against the table.
2. Unload the data by using the DSNTIAUL sample unload program.
3. Drop the table (or table space, if partitioned).
4. Create the ASCII table.

   If you are using the EDITPROC specification for the dropped EBCDIC tables, specify the CREATE TABLE statement with CCSID ASCII and without EDITPROC.
5. Reload the data.
6. Rebind plans and packages.

Altering Indexes

The ALTER INDEX statement allows you to change index attributes with these clauses:

- BUFFERPOOL. Allows you to name the buffer pool to be associated with the index.
- CLOSE. Allows you to specify the priority DB2 should use when determining which open data sets to close.
- CONVERT TO TYPE. Allows you to change index type to either type 1 or type 2.
- DSETPASS. Allows you to specify a password that is passes to VSAM when the data sets of the index are used by DB2. You must also use VSAM access method services to change the data set passwords. Be cautious about changing passwords: ensure that the index space is stopped and no activity is pending on it.
- PART. Allows you to identify a partition of the index.
- FREEPAGE. Allows you to specify how often to leave a page of free space when the index or partition is loaded or reorganized.
- PCTFREE. Allows you to specify the amount of free space to leave in each nonleaf page and subpage when the index or partition is loaded or reorganized.
- USING. As with table spaces, you can change from one group of data sets to another, or between user-managed and DB2-managed data sets, by changing the value of VCAT or STOGROUP.
- PRIQTY. Allows you to specify the primary space allocation for a data set of the index or partition.
- SECQTY. Allows you to specify the secondary space allocation for a data set of the index or partition.
- ERASE. Allows you to specify whether the contents of a data set for the index or partition are erased when the index is dropped.

You can change the options for PRIQTY, SECQTY, and ERASE, in either the same or a new storage group. You can change the storage assignment for a partition of the index by using the PART clause.

- GBPCACHE. Allows you to specifiy what index pages are written to the group bufferpool in a data sharing environment. In a non-data-sharing environment, you can specify this option, but it is ignored.
- PIECESIZE. Allows you to indicate how large DB2 should make the data sets that make up a nonpartitioning index.

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 2-123, 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.

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 would 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 could not take advantage of the new index.

Altering Views

In many cases, changing user requirements can be satisfied by modifying an existing view. But there is no ALTER statement for views; 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 will get an error if you did not re-create the view.

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

SYSIBM.SYSPLANDEP To see what application plans are dependent on the view

SYSIBM.SYSPACKDEP To see what packages are dependent on the view

SYSIBM.SYSVIEWDEP To see what views are dependent on the view

SYSIBM.SYSTABAUTH To see what users are authorized to use the view.

For more information about defining and dropping views, see “Implementing Your Views” on page 2-105.

Changing Data Set Passwords

To change a data set password, follow these steps:

1. Stop the database, using the -STOP DATABASE command.
2. Change the password on all associated data sets by using VSAM access method services.
3. Change the password, using the DSETPASS clause of the ALTER TABLESPACE statement or the ALTER INDEX statement. You must use the same password for all data sets for the same table space or index space.
4. Start the database again, using the -START DATABASE command.

Changing the High-Level Qualifier for DB2 Data Sets

The high-level qualifier for DB2 data sets is the integrated catalog facility catalog name. (From now on, we refer to the integrated catalog facility catalog as the "integrated 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 2-147.

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

- "Define a New Integrated Catalog Alias."
- "Change the Qualifier for System Data Sets" on page 2-140, which includes the DB2 catalog, directory, active and archive logs, and the BSDS.
- "Change Qualifiers for Other Databases and User Data Sets" on page 2-143, which includes the temporary database (DSNDB07) and the default database (DSNDB04), as well as other DB2 databases and user databases.

To concentrate on DB2-related issues, this procedure assumes that the catalog alias resides in the *same* integrated user catalog as that 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 19.

Table 19. 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	DSNZPARM NAME	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 set and system passwords into the BSDS and creates a new DSNHDECP member. You do not have 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 have 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 *Command Reference* and " Section 4. Operation and Recovery" on page 4-1 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 on.

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 Section 3 of *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 Section 3 of *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 integrated 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 2-69). 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:

```
ALTER oldcat.DSNDBC.DSNDB01.*.I0001.A001  -
   NEWNAME (newcat.DSNDBC.DSNDB01.*.I0001.A001)
ALTER oldcat.DSNDBD.DSNDB01.*.I0001.A001  -
   NEWNAME (newcat.DSNDBD.DSNDB01.*.I0001.A001)
ALTER oldcat.DSNDBC.DSNDB06.*.I0001.A001  -
   NEWNAME (newcat.DSNDBC.DSNDB06.*.I0001.A001)
ALTER oldcat.DSNDBD.DSNDB06.*.I0001.A001  -
   NEWNAME (newcat.DSNDBD.DSNDB06.*.I0001.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 also delete the entries from the BSDS at this time, but it is not necessary as they will gradually roll off.

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 if there are 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, do the following:

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 2-141, 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 only the databases in the following list which apply to your environment.

DSNDB07 (temporary 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"
- "Changing User-Managed Objects to Use the New Qualifier" on page 2-145
- "Changing DB2-Managed Objects to Use the New Qualifier" on page 2-145

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 lowest level name of A002, A003, and so on).

Changing Your Work Database to Use the New High-Level Qualifier

There are two methods of changing 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:

   ```
   ALTER oldcat.DSNDBC.DSNDB07.DSN4K01.I0001.A001
      NEWNAME newcat.DSNDBC.DSNDB07.DSN4K01.I0001.A001
   ALTER oldcat.DSNDBC.DSNDB07.DSN32K01.I0001.A001
      NEWNAME newcat.DSNDBC.DSNDB07.DSN32K01.I0001.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 DSNC510;
   ```

```
CREATE TABLESPACE DSN32K01
  IN DSNDB07
  BUFFERPOOL BP32K
  CLOSE NO
  USING VCAT DSNC510;
```

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:

   ```
   ALTER oldcat.DSNDBC.dbname.*.I0001.A001  -
      NEWNAME newcat.DSNDBC.dbname.*.I0001.A001
   ALTER oldcat.DSNDBD.dbname.*.I0001.A001  -
      NEWNAME newcat.DSNDBD.dbname.*.I0001.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)
   ```

   Then, use SQL to verify that you can access the data.

The data set renames 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 will be changed.

Changing DB2-Managed Objects to Use the New Qualifier

Use this procedure when you want to keep the existing DB2 storage group, changing just the high-level qualifier. If you want to move the data to a *new* DB2 storage group, see 2-150.

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 the data sets to user-managed 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:

   ```
   ALTER oldcat.DSNDBC.dbname.*.I0001.A001  -
      NEWNAME newcat.DSNDBC.dbname.*.I0001.A001
   ALTER oldcat.DSNDBD.dbname.*.I0001.A001  -
      NEWNAME newcat.DSNDBD.dbname.*.I0001.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 *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)
   ```

   Then, use SQL to verify that you can access the data.

Moving DB2 Data

We consider the following categories of operations for moving data:

- "Moving a DB2 Data Set" on page 2-149 describes moving a data set from one volume to another
- "Copying a Relational Database" on page 2-150 describes copying a user-managed relational database, with its object definitions as well as its data, from one DB2 subsystem to another, whether on the same MVS system or not.
- "Copying an Entire DB2 Subsystem" on page 2-150 describes copying a DB2 subsystem from one MVS system to another. The copy must include the following:
  - 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 facility catalog that records all the DB2 data sets.

Introduction: Tools Available

Important: Before copying any DB2 data, resolve any inconsistent data 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.

These tools are shipped as parts of DB2:

- The REORG and LOAD utilities.

  Those can be used to move data sets from one DASD device type to another within the same DB2 subsystem. For instructions on using LOAD and REORG, see Section 2 of *Utility Guide and Reference*.
- The COPY and RECOVER utilities. Using those utilities, you can recover an image copy of a DB2 table space onto another DASD device within the same subsystem. For instructions on using COPY and RECOVER, see Section 2 of *Utility Guide and Reference*.

- The DSNTIAUL sample program. The program unloads a DB2 table into a sequential file and generates statements to allow the LOAD utility to load it elsewhere. For instructions on using DSNTIAUL, see Section 2 of *Installation Guide*.
- 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 Section 3 of *Utility Guide and Reference*.

These tools are not parts of DB2 but are separate program products or offerings:

- DataPropagator Relational. This licensed program can extract data from DB2 tables, as well as from DL/I databases, VSAM files, and sequential files. For instructions, see "Loading Data from DL/I" on page 2-116.
- DFSMS/MVS, which contains the following functional components:
  - Data Set Services (DFSMSdss)

    Use DFSMSdss to copy data between DASD devices. For instructions, see *Data Facility Data Set Services: User's Guide and Reference*. You can use on-line panels to control this, through the Interactive Storage Management Facility (ISMF) available with DFSMS/MVS; for instructions, refer to *DFSMS/MVS: Storage Administration Reference for DFSMSdfp*.
  - Data Facilities (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 DASD device type to another within the same DB2 subsystem. Do not migrate the DB2 directory, DB2 catalog, and the temporary 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 DASD volume from one device type to another. The program can be controlled using on-line 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 20 (Page 1 of 2). Tools Applicable to Data-Moving Operations

TOOL	Moving a Data Set	Copying a Data Base	Copying an Entire Subsystem
REORG and LOAD	Yes	-	-
COPY and RECOVER	Yes	-	-
DSNTIAUL	Yes	Yes	-
DSN1COPY	Yes	Yes	-

Table 20 (Page 2 of 2). Tools Applicable to Data-Moving Operations

TOOL	Moving a Data Set	Copying a Data Base	Copying an Entire Subsystem
DXT or DataRefresher	Yes	Yes	-
DFSMSdss	Yes	-	Yes
DFSMSdfp	Yes	-	Yes
DFSMShsm	Yes	-	-

Some of the tools listed rebuild the table space and index space data sets, and therefore generally require longer to execute than the tools that merely copy them. The tools that rebuild are REORG and LOAD, COPY and RECOVER, DSNTIAUL, and DXT. 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 per 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.

Though less efficient in execution, RECOVER might be easy to set up, if image copies and recover jobs already exist. You might need only 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.

Procedures for Moving Data

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 2-124. If you change the integrated catalog password known to DB2 in redefining the storage group, you must execute access method services to change that password in the integrated catalog.

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 or RECOVER: 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.

   ```
   ALTER oldcat.DSNDBC.dbname.*.I0001.A001  -
      NEWNAME newcat.DSNDBC.dbname.*.I0001.A001
   ALTER oldcat.DSNDBD.dbname.*.I0001.A001  -
      NEWNAME newcat.DSNDBD.dbname.*.I0001.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.

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.

Section 3. Security and Auditing

Chapter 3-1. 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 determines 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. If you want first a brief overview of the range of objects that DB2 protects, look at the left-hand columns of Table 21 on page 3-15.

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

Read carefully the introductory section on “Controlling Data Access” on page 3-7. Then skim chapters “Chapter 3-2. Controlling Access to DB2 Objects” on page 3-13 through “Chapter 3-6. Auditing Concerns” on page 3-119. Those chapters describe the tools you use to implement your plan, but they probably contain more detail than you want on a first reading. Read the case study in “Chapter 3-7. A Sample Security Plan for Employee Data” on page 3-135. The sample plan describes decisions of the kind you must make about access to your own data.

Now list your security objectives and the means you will use to achieve them. 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 5 for security and auditing.

Changes for Distributed Processing

- Support for RACF PassTickets lets a requester attach to DB2 without having to flow passwords on the wire. Also, because passwords do not have to be stored in a requesting DB2's communications database, password maintenance is simpler and more secure. See “Sending RACF PassTickets” on page 3-92 for more information.
- Ability to use RACF checks for stored procedures that access non-DB2 resources. Also, you can control which address spaces are allowed to identify themselves to Workload Manager as server address spaces. See “Establishing RACF Protection for Stored Procedures” on page 3-104.
- DB2 for OS/390's support for Distributed Computing Environment (DCE) security as a server lets a DRDA requester connect securely to DB2 by means of a unique global identity. See “Establishing DCE Security for DB2” on page 3-106.
- A new install option, EXTENDED SECURITY, allows you to have DB2 send the requester detailed error codes for the cause of a security failure, as described in Figure 60 on page 3-77 and Figure 61 on page 3-83. This option also allows client programs to allow users to change expired RACF passwords, if those clients wish to implement that function. See “Allowing Users to Change Expired Passwords” on page 3-72.

Changes for Performance

- You can cache package authorizations. See “Caching Authorization IDs for Best Performance” on page 3-27.
- You can cache prepared, dynamic statements. When a prepared dynamic statement has been cached for an authorization ID, subsequent executions of this statement do no require a catalog lookup to check authorization. See Section 6 of *Application Programming and SQL Guide* for details.

Changes for Usability

- Users with installation SYSOPR authority can now issue the START DATABASE command to remove GRECP or LPL status. Previously, installation SYSADM authority was required.
- Programs using the Recoverable Resource Manager Services attachment facility can reuse threads by setting a new sign-on ID. See Section 6 of *Application Programming and SQL Guide* for more information.
- A new system privilege, CREATETMTAB, allows you to define a temporary table. See Table 21 on page 3-15.
- The REFERENCES privilege is refined so that now the privilege can be granted on a specific list of columns in the table. See the description of the GRANT statement in Chapter 6 of *SQL Reference* for more information.
- DB2 provides an installation-wide exit point which lets your own exit routine or another security system, such as the Security Server with OS/390 Release 4, control DB2 authorization. This makes it easier to provide a single point of control for DB2 authorization. For information about Security Server support, see *OS/390 Security Server (RACF) Security Administrator's Guide*. For more information about the DB2 exit point, see “Access Control Authorization Exit” on page X-34.

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 “ Section 1. Introduction” on page 1-1, 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 3-2. Controlling Access to DB2 Objects” on page 3-13. Read also “Chapter 3-6. Auditing Concerns” on page 3-119.

Controlling Data Access

At this point, we need a term to remind you that not every access to data is from a person engaged in an interactive terminal session. For example, it 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 51 on page 3-8 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. There are three types of identifiers: primary authorization IDs, secondary authorization IDs, and SQL IDs.

- Generally it is the primary authorization ID that identifies a 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.

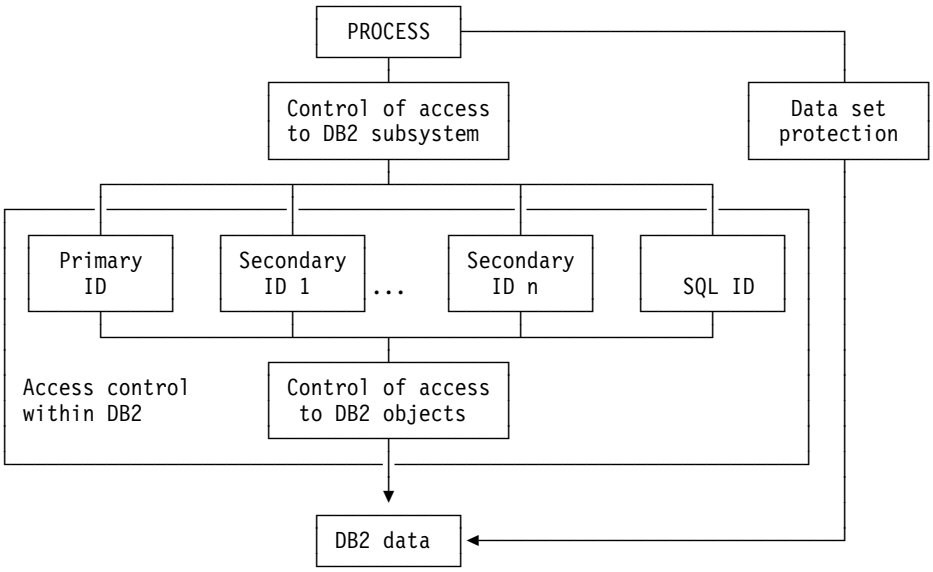


Figure 51. DB2 Data Access Control

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; neither does it know that you are using someone else's ID. DB2 recognizes only the IDs. Therefore, we speak, however awkwardly, of an ID “owning 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. “Implementing Your Views” on page 2-105 shows a typical view, of the sample employee and department tables; it reveals only the employee numbers and names of the managers of a restricted list of departments. 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 3-2. Controlling Access to DB2 Objects" on page 3-13 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 X-34.

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 logged 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 3-4. Controlling Access to a DB2 Subsystem” on page 3-63 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, or you can use Distributed Computing Environment (DCE) security services to authenticate a user.

RACF can:

- Verify an identifier associated with a remote attach 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 3-92 contains information about RACF PassTickets.

If you use DCE, a user is identified by a DCE name (known as a *DCE principal name*) and is authenticated by means of an encrypted DCE ticket obtained from DCE by the client system. The server validates the ticket by invoking a DCE security service and maps the authenticated user DCE identity to a local RACF identity by means of a RACF service. The local RACF ID is used for subsequent authority checking. See “Establishing DCE Security for DB2” on page 3-106 for more information.

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 3-84 for more information. See “The Communications Database for the Server” on page 3-72 for information about controls on the server side.

Data Set Protection

The data in a DB2 subsystem is contained in data sets. As Figure 51 on page 3-8 suggests, those data sets could conceivably be accessed without going through DB2 at all. If the data is at all sensitive, you want to control that route.

If you are using RACF or some similar security system to control access to DB2, then 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 you are not using RACF, you probably want to protect the data sets with VSAM passwords. “Chapter 3-5. Protecting Data Sets” on page 3-113 gives some detailed suggestions.

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 that, 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 3-2. 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 xi.

DB2 controls access to its objects by a set of *privileges*. Each privilege allows an action on some object. Figure 52 shows that there are three primary ways within DB2 to give an ID access to data.[1]

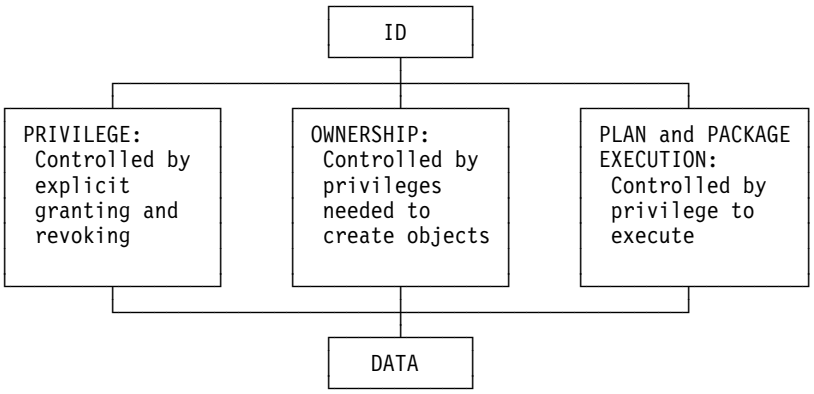


Figure 52. Access to Data within DB2

The security plan must be aware of every way to allow access to data. To write such a plan, first see:

“Explicit Privileges and Authorities” on page 3-14
“Implicit Privileges of Ownership” on page 3-22
“Privileges Exercised through a Plan or a Package” on page 3-24.

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 3-30.

At that point, you have the tools needed for a security plan. Before you begin it, see what others have done in:

“Some Role Models” on page 3-34
“Examples of Granting and Revoking Privileges” on page 3-35.

Privileges granted 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 3-44.

The types of objects to which access is controlled are described in “Chapter 1-2. System Planning Concepts” on page 1-21.

[1] Certain authorities are assigned when DB2 is installed, and can be reassigned by changing the subsystem parameter (DSNZPARM); that could be considered a fourth way to grant 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 function, 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 3-4. Controlling Access to a DB2 Subsystem” on page 3-63; see especially Table 40 on page 3-65 and Table 41 on page 3-66.

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 3-71, 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. For convenience, we group them into privileges over tables, plans, packages, collections, databases, the entire subsystem, and uses of objects. The following tables list and describe explicit privileges.

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 a list of columns in the table
SELECT	SELECT, to retrieve data from the table
UPDATE	UPDATE, to update all columns or a specific list of columns [2]

GRANT ALL on a table can grant all table privileges listed above.

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. Explicit DB2 Package Privileges

Package Privilege	Allows These Functions 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 on a package can grant all package privileges listed above.

Table 24. Explicit DB2 Collection Privileges

Collection Privileges	Allows These Functions 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. Explicit DB2 Database Privileges

Database Privileges	Allow These Functions 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 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. Explicit DB2 System Privileges

System Privileges	Allow These Functions
ARCHIVE	The ARCHIVE LOG command, to archive the current active log, the DISPLAY ARCHIVE command, to give information about input archive logs, 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
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
CREATESG	The CREATE STOGROUP statement, to create a storage group
CREATETMTAB	The CREATE GLOBAL TEMPORARY TABLE statement, to define a temporary table
DISPLAY	The DISPLAY ARCHIVE, DISPLAY BUFFERPOOL, DISPLAY DATABASE, DISPLAY LOCATION, 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. Explicit DB2 Use Privileges

Use Privileges	Allows the Use of These Objects
USE OF BUFFERPOOL	A buffer pool
USE OF STOGROUP	A storage group
USE OF TABLESPACE	A table space

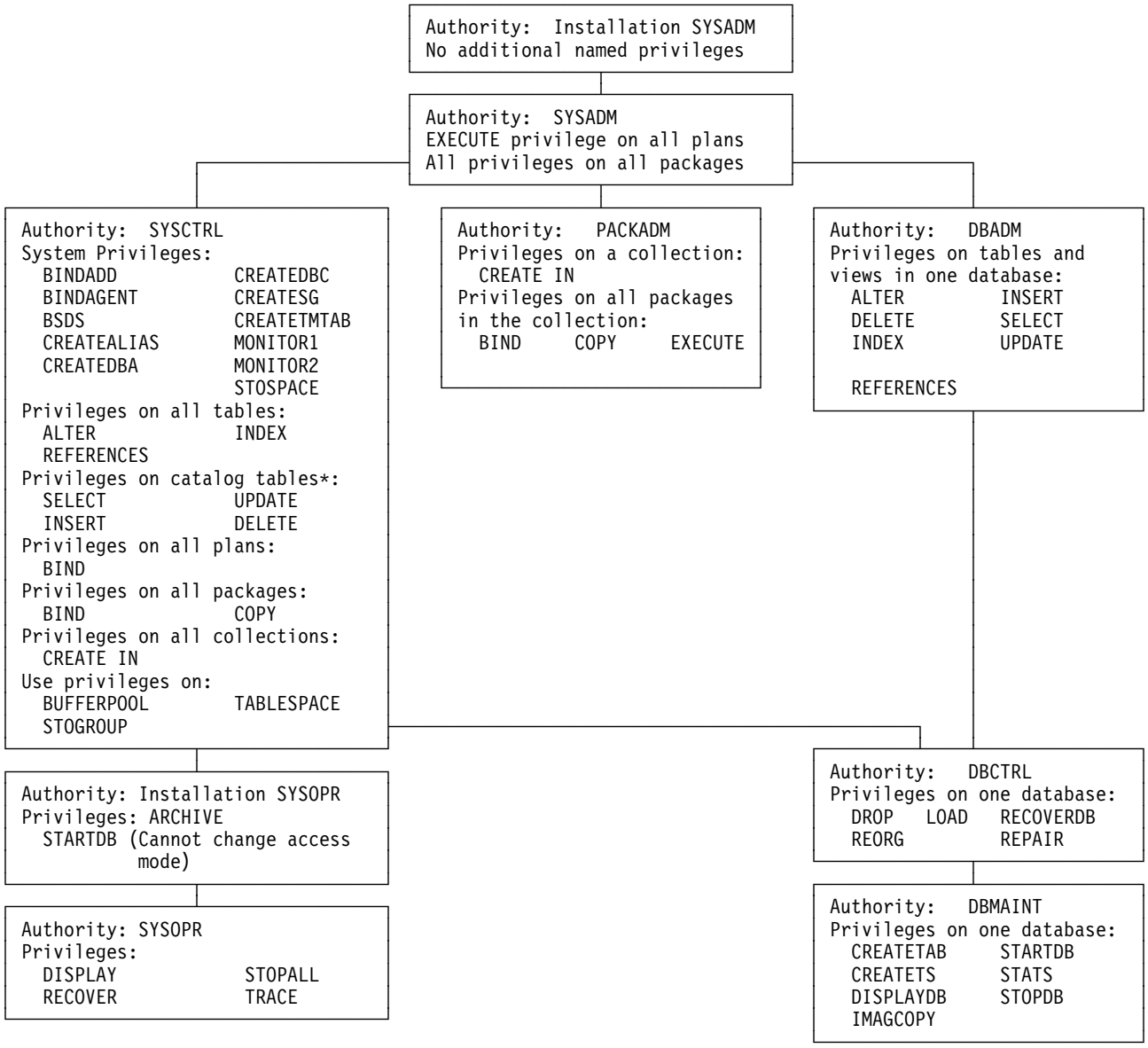
Privileges Needed for Statements, Commands, and Utility Jobs: For lists of all privileges and authorities that let you:

- Execute a particular SQL statement, see the description of the statement in Chapter 6 of *SQL Reference*.
- Issue a particular DB2 command, see the description of the command in Chapter 2 of *Command Reference*.

- Run a particular type of utility job, see the description of the utility in *Command Reference*.

Administrative Authorities

Figure 53 shows how privileges are grouped into authorities and how the authorities form a branched hierarchy. Table 28 on page 3-19 expands on the figure to include capabilities of each authority that are not represented by explicit privileges described in Table 21 on page 3-15.



* For the applicable catalog tables and the operations that can be performed on them by SYSCTRL, see the DB2 catalog appendix in the SQL Reference.

Figure 53. 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 28 (Page 1 of 3). DB2 Authorities

Authority	Description
SYSOPR	System operator: • Can issue most DB2 commands • *Cannot* issue ARCHIVE LOG, START DATABASE, STOP DATABASE, and RECOVER BSDS • Can terminate any utility job • Can run the DSN1SDMP utility.
Installation SYSOPR	One or two IDs are named to this when DB2 is installed. They have all the privileges of SYSOPR, plus: • Authority is *not* recorded in the DB2 catalog. The catalog need not be available to check installation SYSOPR authority. • No ID can revoke the authority; it can be removed only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM). Those IDs can also: • Access DB2 when the subsystem is started with ACCESS(MAINT). • Run all allowable utilities on the directory and catalog databases (DSNDB01 and DSNDB06). • Run the REPAIR utility with the DBD statement. • Start and stop the database containing the application and object registration tables (ART and ORT). • Issue dynamic SQL statements not controlled by the DB2 governor. • Issue a START DATABASE command to recover objects that have LPL entries or group recovery pending status. These IDs cannot change the access mode.
PACKADM	Package administrator: 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, can grant those privileges to others. If the installation option BIND NEW PACKAGE is BIND, also has the privilege to add new packages or new versions of existing packages.
DBMAINT	Database maintenance: In a specific database, the holder can create certain objects, run certain utilities, and issue certain commands. If held with the GRANT option, 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: includes DBMAINT over a specific database, plus the database privileges to run utilities that can change the data. If held with the GRANT option, can grant those privileges to others.
DBADM	Database administration: includes DBCTRL over a specific database, plus privileges to access any of its tables through SQL statements. If held with the GRANT option, can grant those privileges to others. Can also drop and alter any table space, table, or index in the database and issue a COMMENT ON, LABEL ON, or LOCK TABLE statement for any table. Has no privileges on views owned by other IDs, even if they are based on tables in the database.

Table 28 (Page 2 of 3). DB2 Authorities

Authority	Description
SYSCTRL	System control: 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, it can: • Act as installation SYSOPR (when the catalog is available) or DBCTRL over any database • Run any allowable utility on any database • Issue a COMMENT ON, LABEL ON, or LOCK TABLE statement for any table • Create a view for itself or others on any catalog table • Create tables and aliases for itself or others • Bind a new plan or package, naming any ID as the owner. Without additional privileges, it *cannot*: • Execute DML statements on user tables or views • Run plans or packages • Set the current SQL ID to a value that is not one of its primary or secondary IDs • Start or stop the database containing the ART and ORT • Act fully as SYSADM or as DBADM over any database • Access DB2 when the subsystem is started with ACCESS(MAINT). 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.
SYSADM	System administrator: includes SYSCTRL, plus access to all data. It can: • Use all the privileges of DBADM over any database • Use EXECUTE and BIND on any plan or package, COPY on any package • Use privileges over views owned by others • Set the current SQL ID to any valid value, whether or not it is currently a primary or secondary authorization ID • Create and drop synonyms and views for others on any table • Use any valid value for OWNER in BIND or REBIND • Drop database DSNDB07. • Grant any of the privileges listed above to others. Holders 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 cannot specifically grant those privileges.

Table 28 (Page 3 of 3). DB2 Authorities

Authority	Description
Installation SYSADM	One or two IDs are named to this when DB2 is installed. They have all the privileges of SYSADM, plus: • 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.) • No ID can revoke the authority; it can be removed only by changing the module that contains the subsystem initialization parameters (typically DSNZPARM). Those IDs can also: • Run the CATMAINT utility. • Access DB2 when the subsystem is started with ACCESS(MAINT). • Start databases DSNDB01 and DSNDB06 when those are stopped or in restricted status. • Run the DIAGNOSE utility with the WAIT statement. • Start and stop the database containing the ART and ORT.

Field-level Access Control by Views

Any of the table privileges, except ALTER, REFERENCES and INDEX, can also be granted on a view. By creating a view and granting privileges on it, it is possible to give an ID access only to 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 DSN8510.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. DBADM on DSNDB06 can update SYSIBM.SYSPROCEDURES for stored procedures.

Authorities 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 the utilities listed in Table 29 on DSNDB06, but not by granting privileges on DSNDB01 directly.

Every authority, except SYSOPR, carries the privilege to run some utilities on databases DSNDB01 and DSNDB06. Table 29 shows what utilities the other authorities can run on those databases.

Table 29. Utility Privileges on the DB2 Catalog and Directory

Utilities	**Authorities**		
	Installation SYSOPR, SYSCTRL, SYSADM, Installation SYSADM	**DBCTRL, DBADM on DSNDB06**	**DBMAINT on DSNDB06**
LOAD, REPAIR DBD	None (cannot be run on DSNDB01 and DSNDB06)		
Note: LOAD can be used to add lines to SYSIBM.SYSSTRINGS.			
CHECK DATA, REORG TABLESPACE, STOSPACE	Yes	No	No
Note: CHECK DATA can be run only by an ID with installation SYSOPR authority, and cannot be run on DSNDB01.			
RECOVER INDEX, RECOVER TABLESPACE, REORG INDEX, REPAIR, REPORT	Yes	Yes	No
CHECK INDEX, COPY, MERGECOPY, MODIFY, QUIESCE, RUNSTATS	Yes	Yes	Yes

Implicit Privileges of Ownership

You create DB2 objects, except for plans and packages, by issuing SQL CREATE statements in which you name the object. The name can be *unqualified* (for example, T1) or, for tables, views, indexes, aliases, and synonyms, *qualified* (for example, GAMMA.T1). When you create the 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 3-24.)

Establishing Ownership of Objects with Unqualified Names

If the name is unqualified, then you establish the object's ownership in these ways:

- If you issue the CREATE statement dynamically, perhaps using SPUFI, QMF, or some similar program, then the owner of the object created is your current SQL ID. That ID must have the privileges needed to create the object.
- If you issue the CREATE statement statically, by running a plan or package that contains it, then the ownership of the object created depends on an option of the bind operation. The QUALIFIER option allows the binder to name a qualifier to be used for all unqualified names of tables, views, indexes, aliases, or synonyms that appear in the plan or package. (The owner of the plan or package must have all required privileges on the objects designated by the qualified names.) If the QUALIFIER option is not used, then the ID named by the OWNER option, or the default owner, is used as the qualifier.

Establishing Ownership of Objects with Qualified Names

If you create a table, a view, an index, an alias, or a synonym with a qualified name, then the qualifier becomes the owner of the object, subject to these restrictions:

- If you issue the CREATE statement dynamically, and have no administrative authority, then 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, and if it has SYSADM or SYSCTRL authority, you can also use any qualifier for a view or alias.

  If the current SQL ID has at least DBCTRL authority, then the qualifier ID does not need any privileges. In addition to DBCTRL authority, the SQL ID must have any additional privileges 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, then 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, then the qualifier can only be 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, and if the owner has SYSADM or SYSCTRL authority, it can also use any qualifier for a view or an alias.

Privileges by Type of Object

The table below lists implicit privileges of ownership by type of object.

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) exercised in creating 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

Object type	Implicit privileges of ownership
Table	To alter or drop the table or any indexes on it To lock the table, comment on it, or label it To create an index or view for the table To select or update any row or column To insert or delete any row To use the LOAD utility for the table To define referential constraints on any table or set of columns
Index	To alter or drop the index
View	To drop the view, comment on or label it, select any row or column 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

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

There is no way to revoke the privileges implicit in ownership. Except for a plan or package, as long as an object exists, there is no way to change its owner. 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]

In practice, however, it can be desirable to share the privileges of ownership. That can be done by making the owning ID a secondary ID to which several primary authorization IDs are connected. The list of primary IDs connected to the secondary ID can be changed without having to drop and re-create the object.

Privileges Exercised through a Plan or a Package

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 privilege required. 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 used in validating the plan or package, but only within the restrictions imposed by the SQL statements in the original program.

For example, the program might contain:

```
EXEC SQL
  SELECT * INTO :EMPREC FROM DSN8510.EMP
    WHERE EMPNO='000010';
```

[3] Dropping a package does not delete all privileges on it if another version of the package still remains in the catalog.

The example puts the data for employee number 000010 into the host structure EMPREC. The data comes from table DSN8510.EMP. But the ID that has EXECUTE privilege for this plan can only access rows in the DSN8510.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, then 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 one, use the OWNER option to name the owner of the resulting plan or package.

- 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 REBIND.

If you omit the OWNER option:

- On BIND, your primary ID becomes the owner.
- On REBIND, the previous owner retains ownership.

Not all systems that can bind a package at a DB2 system 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 or view names. The default qualifier for those names in a plan or package is the owner of the plan or package. But, you can use the QUALIFIER option of BIND to specify a different qualifier.

Using the BINDAGENT privilege and the OWNER and QUALIFIER options, you have considerable flexibility in performing bind operations. For example, if ALPHA has the BINDAGENT privilege from BETA, and BETA has privileges over tables owned by GAMMA, then ALPHA can bind a plan using OWNER (BETA) and QUALIFIER (GAMMA). ALPHA, as merely a binding agent, need have no privileges over the tables and does not have the privilege to execute the plan.

Checking Authorization to Execute

The plan or package owner must have authorization to execute all SQL statements embedded in the plan or package. But you do not need to have the authorizations in place when the plan or package is bound; in fact, the SQL objects referred to do not even have to exist at that time.

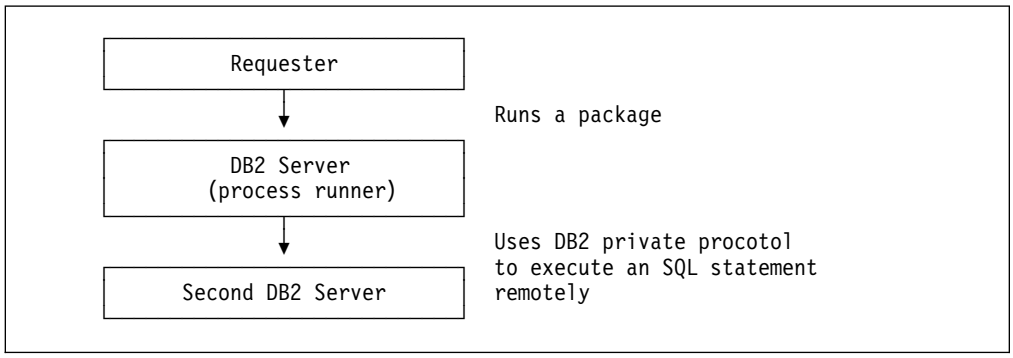
A bind operation always checks whether a local object exists and whether the owner has the required privileges on it. Any failure results in a message. To choose whether the failure prevents the bind operation from completing, use the VALIDATE option of BIND PLAN and BIND PACKAGE, and also the SQLERROR option of BIND PACKAGE. See Section 5 of *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 31 on page 3-30 shows which IDs can supply the required authorizations for different types of statement.

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 3-27 to
perform authorization checking.

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 system-directed access. See Figure 54



| *Figure 54. Execution at a Second DB2 Server*

In the figure, a remote requester, either a DB2 for OS/390 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 server. The ID that is checked for the privileges needed to run at the second server can be the owner of the plan running at the requester (if the requester is DB2 for MVS/ESA or DB2 for OS/390), the owner of the package running at the DB2 server, or the authorization ID of the process that runs the package at the first DB2 server (the "process runner"). Which ID is used depends on these four factors:

- Whether the requester is DB2 for OS/390 or DB2 for MVS/ESA, or a different system.
- The value of the bind option DYNAMICRULES (RUN or BIND). See Chapter 2 of *Command Reference* for information about DYNAMICRULES.
- Whether the parameter HOPAUTH at the DB2 server site was set to YES or NO when running the installation job DSNTIJUZ. The default value is YES.
- Whether the statement executed at the second server is static or dynamic SQL.

Table 30 on page 3-27 shows how these factors determine the ID that must hold the required privileges.

Table 30. The Authorization ID That Must Hold Required Privileges

Requester	DYNAMICRULES	HOPAUTH	Statement	Authorization ID
DB2 for MVS/ESA or DB2 for OS/390	RUN (default)	n/a	Static	Plan owner
			Dynamic	Process runner
	BIND	n/a	Either	Plan owner
# Different System # or # RRSAF application # without a plan	RUN (default)	YES (default)	Static	Package owner
			Dynamic	Process runner
		NO	Either	Process runner
	BIND	n/a	Either	Package owner

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 or DB2 for MVS/ESA system and is accessed using DRDA
- At a DB2 for OS/390 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 or DB2 for MVS/ESA system. Table 30 lists the specific privileges.

Caching Authorization IDs for Best Performance

You can specify that DB2 cache authorization IDs for plans or packages. Having IDs cached can greatly improve performance, especially when user IDs are reused frequently. There is a cache for each plan, and one global cache for packages. For a data sharing group, each member does its own authorization caching.

Caching IDs for Plans: 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 Section 5 of *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 the following:

- 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.** The benefit is also provided for use of *.* or *.package-id*.

If you do not use collections, or if you have all your packages in the same collection, you will see less performance benefit than if you have frequently-run packages in different collections.

Set the size of the package authorization cache using the PACKAGE AUTH CACHE field on installation panel DSNTIPP. The default value, 32K, is enough storage to support about 400 *collection-id.package-id* entries or *collection-id.** entries. For each increase of 100 packages, add 8KB of storage to the cache. The *ssnm*DBM1 address space provides the storage.

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 tells you how often DB2 was successful at reading package authorization information from the cache.

Authorization for Stored Procedures

The privileges required to execute a plan or package containing a CALL statement must include at least one of the following:

- The EXECUTE privilege on the plan or package
- Ownership of the plan or package
- PACKADM authority for the collection (packages only)
- SYSADM authority

Additional privileges are required on each package used by the stored procedure during its execution. The application server determines the privileges that are required and the authorization ID that must have the privileges. See Chapter 6 of *SQL Reference* for more information about authorization for the CALL statement.

Controls in the Program

Because an ID executes a package or an application plan by running a program, it is sometimes useful to implement control measures in the program. For example, consider the SQL statement on page 3-24, that 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 so 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 Caution about Controls in the Program

We do not recommend using programs to extend security. Whenever possible, other techniques, such as stored procedures or views, should be used 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, then it is important to 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 *Command Reference*.

Privileges Required for Remote Packages

Generally, the privileges 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 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.

[4] Or BIND, depending on the installation option BIND NEW PACKAGE.

- 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 the EXECUTE privilege for the package at the DB2 server.
- For DRDA, if the server is a DB2 for OS/390, the authorization ID of the process (primary or any secondary) must have the EXECUTE privilege for the package at the DB2 server.
- If the server is not DB2 for OS/390, the primary authorization ID must have whatever privileges are needed. Check that product's documentation.

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. The following table summarizes, for different actions, which IDs can provide the necessary privileges. For more specific details on any statement or command, see *SQL Reference* or *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 slower the check proceeds. For dynamic SQL, the current SQL ID is checked first; the operation is quickest if that ID has all the necessary privileges.

Authorization for Dynamic SQL Statements

If the value of the bind option DYNAMICRULES is RUN, the authorization ID for dynamic SQL statements is determined at run time. The IDs that must hold required privileges are the primary ID and its associated secondary IDs, and one of those IDs is the current SQLID. The current SQLID holds the privileges exercised when issuing certain dynamic SQL statements.

If the value of DYNAMICRULES is BIND, the authorization ID for dynamic SQL statements is determined at bind time. The ID that must hold the required privileges is the plan or package owner. Neither secondary IDs associated with that ID nor the current SQLID are checked for authorization. Hence, users issuing dynamic statements through the plan or package can exercise all the privileges of the owner and only those privileges. See Chapter 2 of *Command Reference* for additional information about DYNAMICRULES.

Table 31 (Page 1 of 4). Privileges Required for Basic Operations

Operation	ID	Required Privileges
Dynamic SQL Statements		

Table 31 (Page 2 of 4). Privileges Required for Basic Operations

Operation	ID	Required Privileges
GRANT	Current SQL ID	Any of these: The applicable privilege with the grant option An authority that includes the privilege, with the grant option (not needed for SYSADM or SYSCTRL) Ownership that implicitly includes the privilege
REVOKE	Current SQL ID	Must have granted the privilege being revoked, or hold SYSCTRL or SYSADM authority.
CREATE, for unqualified object name	Current SQL ID	Applicable table or database 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 = RUN	All primary and secondary IDs and the current SQL ID together	As required by the statement; see "Composite Privileges" on page 3-33. Unqualified object names are qualified by the value of the special register CURRENT SQLID. See "Authorization for Dynamic SQL Statements."
Other dynamic SQL if DYNAMICRULES = BIND	Plan or package owner	As required by the statement; see "Composite Privileges" on page 3-33. Unqualified object names include the value of QUALIFIER. See "Authorization for Dynamic SQL Statements."
Operations on Plans and Packages		
Execute a plan	Primary or any secondary ID	Any of these: Ownership of the plan EXECUTE privilege for the plan SYSADM authority
Bind embedded SQL statements, for any bind operation	Plan or package owner	Any of these: Applicable privileges required by the statements Authorities that include the privileges Ownership that implicitly includes the privileges Object names include the value of QUALIFIER, where it applies.
Include package in PKLIST	Plan owner	Any of these: Ownership of the package EXECUTE privilege for the package PACKADM authority over the package collection SYSADM authority
BIND a new plan using the default owner or primary authorization ID	Primary ID	BINDADD privilege, or SYSCTRL or SYSADM authority

Table 31 (Page 3 of 4). Privileges Required for Basic Operations

Operation	ID	Required Privileges
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: BINDADD privilege and CREATE IN privilege for the collection PACKADM authority for the collection SYSADM or SYSCTRL authority If BIND NEW PACKAGE is BINDADD, any of these: BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection 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: Ownership of the plan or package BIND privilege for the plan or package BINDAGENT from the plan or package owner PACKADM authority for the collection (for a package only) SYSADM or SYSCTRL authority See also “Multiple Actions in One Statement” on page 3-33.
BIND a new version of a package, with default owner	Primary ID	If BIND NEW PACKAGE is BIND, any of these: BIND privilege on the package or collection BINDADD privilege and CREATE IN privilege for the collection PACKADM authority for the collection SYSADM or SYSCTRL authority If BIND NEW PACKAGE is BINDADD, any of these: BINDADD privilege and either the CREATE IN or PACKADM privilege for the collection SYSADM or SYSCTRL authority
FREE or DROP a package	Primary or any secondary ID	Any of these: Ownership of the package BINDAGENT from the package owner PACKADM authority for the collection SYSADM or SYSCTRL authority
COPY a package	Primary or any secondary ID	Any of these: Ownership of the package COPY privilege for the package BINDAGENT from the package owner PACKADM authority for the collection SYSADM or SYSCTRL authority
FREE a plan	Primary or any secondary ID	Any of these: Ownership of the plan BIND privilege for the plan BINDAGENT from the plan owner SYSADM or SYSCTRL authority

Table 31 (Page 4 of 4). Privileges Required for Basic Operations

Operation	ID	Required Privileges
Name a new OWNER other than the primary authorization ID for any bind operation	Primary or any secondary ID	Any of these: New owner is the primary or any secondary ID BINDAGENT from the new owner SYSADM or SYSCTRL authority

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 no one of your IDs has all the privileges required. If the bind option DYNAMICRULES is RUN, 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 if the DYNAMICRULES option is BIND when you issue the dynamic statement. In either of those cases, all the required privileges must be held by the single ID that owns the plan or package.

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. When someone with FREDDY and REUBEN as secondary authorization IDs issues the 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 32. 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	SYSOPR authority
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 some database; use of storage groups and buffer pools
System Programmer	Installs a DB2 system; recovers the DB2 catalog; repairs data	Installation SYSADM, given 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 tables; 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 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 enables you to grant privileges explicitly. The REVOKE statement enables you to 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. You can also grant a table privilege to any ID anywhere that uses system-directed access to your data, by issuing:

```
GRANT privilege TO PUBLIC AT ALL LOCATIONS;
```

The privilege can be any table privilege except ALTER, INDEX, or REFERENCES.

If you grant a privilege to PUBLIC, the DB2 catalog tables record the grantee as PUBLIC. If you grant to PUBLIC AT ALL LOCATIONS, the grantee is PUBLIC*. PUBLIC is a special identifier used by DB2 internally; it should not be used as a primary or secondary authorization ID. PUBLIC* cannot be used as an ID. When a privilege is revoked from PUBLIC or PUBLIC AT ALL LOCATIONS, authorization IDs to which the privilege was specifically granted still retain it.

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, held by PUBLIC. If PUBLIC loses the privilege, the view is dropped unless the privilege was also granted specifically to USER2.
- Another user binds a plan, PLAN1, whose program refers to the table 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 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 arriving 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 3-71.)

The end result is that the query is associated with a set of IDs known to your local DB2. How you assign privileges to those IDs is no different from how you assign them to IDs associated with queries arising locally.

There are, however, some differences in the privileges that a query using system-directed 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. In order to focus upon GRANT, and not the broader topic of security, the example is one for which the data is not highly critical.

Suppose that the Spiffy Computer Company wants to create a database to hold information that is usually posted on hallway bulletin boards—important 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 needed for this application, the security plan provides for the roles shown in Figure 55.

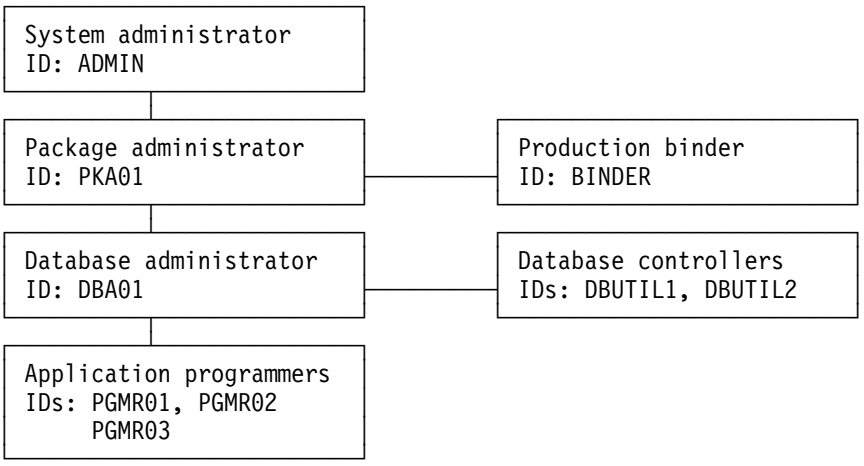


Figure 55. Security Plan for the Spiffy Computer Company. Lines connect the grantor of a privilege or authority to the grantee.

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 pass that privilege on 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 pass that privilege on 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 pass on 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

But the database administrator at Spiffy wants help running the COPY and RECOVER utilities, so he grants DBCTRL authority over database DB1 to DBUTIL1 and DBUTIL2.

```
    User ID:  DBUTIL1, DBUTIL2
  Authority:  DBCTRL over DB1
```

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 their Software Support department to create tables in the DB1 database. He 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 both the CREATETAB and CREATETS privileges, as well as the privileges to use the SG1 storage group and one of the buffer pools, BP0, for the implicitly created table spaces.

```
RACF group ID: DEVGROUP
   Privileges: (All without GRANT)
               CREATETAB on DB1
               CREATETS on DB1
               Use of SG1
               Use of BP0
```

DBA01 owns database DB1 and has the privileges to use storage group SG1 and buffer pool BP0 (both with the GRANT option). He 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 Software Support 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. But all of those resources are controlled by DEVGROUP, which is a RACF group ID.

So it is not necessary to grant privileges over those resources specifically to each of PGMR01, PGMR02, and PGMR03. All that is needed is to connect each ID to the RACF group DEVGROUP. (That plan assumes 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 3-4. Controlling Access to a DB2 Subsystem" on page 3-63.)

```
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 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 contained in T1. Someone must bind the plan and packages that accompany the program, and that requires an additional privilege.

```
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 to be owned by DEVGROUP. Any member of the group can rebind a plan or package owned by DEVGROUP.

Software Support proceeds to create and test the program.

Moving PROGRAM1 into Production

Spiffy has a different set of tables, containing actual data and 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.

```
   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. He does this 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; it is enough that they have the privilege of executing the plan.

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.

That done, the package owner can issue the statement:

```
GRANT EXECUTE ON PACKAGE PROGRAM1 TO PUBLIC;
```

Any system connected to the original DB2 location can then run PROGRAM1 and execute the package, using DRDA access. (If the remote system is another DB2, then there must be a plan bound there that includes the package in its package list.)

That solution, of course, is vastly simplified. Here we are focusing on granting appropriate privileges and authorities. In practice, you would also have 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?

And so on.

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 on 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 it 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. But 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.

[5] DB2 does not cascade a revoke of SYSADM authority from the installation SYSADM authorization IDs.

Privileges Granted from Two or More IDs

In addition to the CREATETAB privilege 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 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 56 diagrams the situation; arrows represent the granting of the CREATETAB privilege.

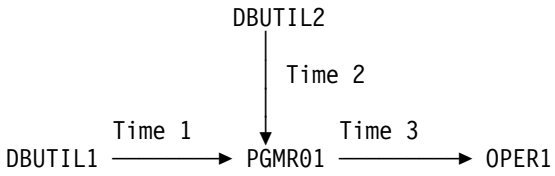


Figure 56. Authorization Granted by Two or More IDs

As in the diagram, suppose that DBUTIL1 and DBUTIL2 at, respectively, Time 1 and Time 2, each issued 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. But 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 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 granted by DBUTIL1. That privilege was revoked, so the privileges dependent on it are revoked.

Revoking Privileges Granted by Other IDs

An ID with SYSADM or SYSCTRL authority can revoke privileges granted by other IDs.

To revoke the CREATETAB privilege on database DB1 from PGMR01 entirely, use:

```
REVOKE CREATETAB ON DATABASE DB1 FROM PGMR01 BY ALL;
```

To revoke privileges granted by DBUTIL1, and leave intact the same privileges if they were granted by any other ID, use:

```
REVOKE CREATETAB, CREATETS ON DATABASE DB1 FROM PGMR01 BY DBUTIL1;
```

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 from all other IDs it was granted to. If the SELECT privilege on the base table is revoked from the owner of the view, the view is dropped. For example, suppose OPER2 has the SELECT and INSERT privileges on table T1 and creates a view of it. 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, the view is dropped.

Views Created by SYSADM: It is possible for 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. But it is still the privilege on the base table that 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.

- If IDADM creates a view for OPER when OPER has enough privileges to create the view by itself, then 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, then 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 held on the base table.

Application Plan and Package Invalidation: If the owner of an application plan or package loses a privilege 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, there is no effect on the plan.

| ***Implications for Caching:*** If authorization is being cached for packages, a revoke
| of EXECUTE authority on the package from an ID causes that ID to be removed
| from the cache.

| If authorization is being 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 execute a full PREPARE. If the plan or
| package is bound with KEEPDYNAMIC(YES), which means the application does
| not have to explicitly re-prepare the statement after a commit operation, this means
| 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
| needed 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. This feature allows you to 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 wish to delete an Install SYSADM, by updating the SYSTEM ADMIN 1 or 2 field on installation panel DSNTIPP.
3. REVOKE SYSADM authority from it 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 held by IDs:

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, and table spaces
SYSIBM.SYSTABAUTH	Tables and views
SYSIBM.SYSUSERAUTH	System authorities.

For descriptions of the columns of each table, see Appendix D of *SQL Reference* . The suggestions that follow show how to extract useful information about privileges.

Retrieving Information in the Catalog

You can query the DB2 catalog tables through SQL SELECT statements. Executing those statements requires appropriate privileges and authorities, and you can control access to the catalog by granting and revoking those. For suggestions about securing the catalog, see “Using Views of the DB2 Catalog Tables” on page 3-47.

The following examples suggest some of the information you can get from the DB2 catalog.

Retrieving All DB2 Authorization IDs with Granted Privileges

Each of the catalog tables listed above includes 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:

```
SELECT GRANTEE, 'PLAN    ' FROM SYSIBM.SYSPLANAUTH
  WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'PACKAGE ' FROM SYSIBM.SYSPACKAUTH
  WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'SYSTEM  ' FROM SYSIBM.SYSUSERAUTH
  WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'DATABASE' FROM SYSIBM.SYSDBAUTH
  WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'TABLE   ' FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'COLUMNS ' FROM SYSIBM.SYSCOLAUTH
  WHERE GRANTEETYPE = ' ' UNION
SELECT GRANTEE, 'USE     ' FROM SYSIBM.SYSRESAUTH
  WHERE GRANTEETYPE= ' ';
```

Periodically, you should compare the list retrieved by that statement 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.) But, you might want authority granted from several different IDs. For example, you might want to retain a privilege if it 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 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 (DSN8510.EMP in database DSN8D51A), issue:

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE TTNAME = 'EMP' AND TCREATOR = 'DSN8510'
  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 = 'DSN8510' 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 = 'DSN8D51A';
```

To retrieve the columns of DSN8510.EMP for which update privileges have been granted on a specific set of columns, issue:

```
SELECT DISTINCT COLNAME, GRANTEE, GRANTEETYPE FROM SYSIBM.SYSCOLAUTH
  WHERE CREATOR='DSN8510' 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 DSN8510.EMP, issue:

```
SELECT DISTINCT GRANTEE
   FROM SYSIBM.SYSTABAUTH
     WHERE TTNAME = 'EMP' AND TCREATOR='DSN8510' AND GRANTEETYPE=' '
     AND UPDATEAUTH <> ' ';
```

The query returns only IDs to which update privileges have been specifically granted. It does not return those who have the privilege because of SYSADM or DBADM authority. They could be included by forming the union with another query.

Retrieving the Tables an ID is Authorized to Access

To retrieve the list of tables and views that PGMR001 can access, issue:

```
SELECT DISTINCT TCREATOR, TTNAME FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEE = 'PGMR001' AND GRANTEETYPE =' ';
```

To retrieve the tables, views, and aliases that PGMR001 owns, issue:

```
SELECT NAME FROM SYSIBM.SYSTABLES
  WHERE CREATOR = 'PGMR001';
```

Retrieving the Plans and Packages That Access a Table

The statement below retrieves the names of application plans and packages that refer to table DSN8510.EMP directly.

```
SELECT DISTINCT GRANTEE FROM SYSIBM.SYSTABAUTH
  WHERE GRANTEETYPE = 'P'       AND
        TCREATOR    = 'DSN8510'  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: it 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 it is not desirable 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 3-3. 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* you can control how specific plans or package collections can use those statements. Figure 57 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 register the objects that the plans and collections are associated with in 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" on page 3-50.

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 3-57.

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 57. 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 application registration table and the object registration table. 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, there are installation options you must choose to make any use of data definition control; see "Required Installation Options." 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 3-51.
  - Registered applications have total control with some exceptions. See "Controlling by Application Name with Exceptions" on page 3-52.
- Control by object name
  - All objects in the system are registered and controlled by name. See "Controlling by Object Name" on page 3-54.
  - Some specific objects are registered and controlled, but with exceptions. DDL is accepted for objects that are not registered. See "Controlling by Object Name with Exceptions" on page 3-56.

The names in some columns in the ART and ORT can be represented by patterns that use the percent sign (%) and the underscore (_) characters. 3-51 tells you how to do this.

Required Installation Options

To make any use of the application and object registration tables, you must install data definition control support. Do that on panel DSNTIPZ by entering:

```
1 INSTALL DD CONTROL SUPT. ===> YES
```

Also on panel DSNTIPZ, choose the names for the registration tables in your DB2 subsystem, their owner, and the database they reside in. You can accept the default names or assign names of your own. The default names are:

```
6 REGISTRATION OWNER       ===> DSNRGCOL
7 REGISTRATION DATABASE    ===> DSNRGFDB
8 APPL REGISTRATION TABLE  ===> DSN_REGISTER_APPL
9 OBJT REGISTRATION TABLE  ===> DSN_REGISTER_OBJT
```

We use those names throughout this chapter. If you choose table names of your own, each name can have a maximum of 17 characters.

Four other options on panel DSNTIPZ determine how DDL statements are controlled. The following sections of this chapter show how specific objectives make use of:

```
 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, enter:

   ```
   CONTROL ALL APPLICATIONS ===> YES
   ```

   That choice allows only package collections or plans registered in the application registration table to use DDL statements. (This case, then, does not require any use of the object registration table.)

2. Register, in the application registration table, 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 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 executing the DDL or, if there is no package, the name of the plan executing the DDL
APPLIDENTTYPE	Type of item named by APPLIDENT: P Application plan C Package collection
DEFAULTAPPL	Whether the plan or package collection named by APPLIDENT can use DDL. Enter **Y** (Yes); the default is N (No).

   (There are other columns in which you can enter information for your own use. For a complete description of the table, see “Columns of the Application Registration Table (ART)” on page 3-57.)

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 33 shows the entries you need in your application registration table.

Table 33. 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 3 of *SQL Reference*. However, there is one difference: 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 _ or %. To use the pound sign (#), enter:

```
5 ART/ORT ESCAPE CHARACTER ===> #
```

With that specification, the pound sign can be used in names here in the same way as an escape character is used in an SQL LIKE predicate.

An Inactive Table Entry: In the table, the row with TR% for APPLIDENT was once entered with the value Y for DEFAULTAPPL, to allow DDL to be executed by any plan with a name beginning with TR. Later, DEFAULTAPPL was 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 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, enter:

   ```
   CONTROL ALL APPLICATIONS ===> NO
   ```

   That choice allows unregistered applications to use DDL statements. The object registration table determines restrictions that apply to that use.

2. Also on panel DSNTIPZ, enter:

   ```
   UNREGISTERED DDL DEFAULT ===> APPL
   ```

   That choice restricts the use of DDL statements for objects *not* registered in the object registration table: 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 object registration table, 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 object named: C Table, view, index, synonym, or alias D Database T Table space S Storage group

APPLMATCHREQ 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 there is no package, the name of the plan that can have exclusive control

APPLIDENTTYPE Type of item named by APPLIDENT:

P Application plan
C Package collection

(There are other columns in which you can enter information for your own use. For a complete description of the table, see "Columns of the Object Registration Table (ORT)" on page 3-58.)

Example: As in the example for "Controlling by Application Name" on page 3-51, 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%). But you also want these specific exceptions:

- Object KIM.VIEW1 can be created, altered, or dropped only by PLANC.
- Object BOB.ALIAS, only by the package collection PACKD.
- Object FENG.TABLE2, by *any* plan or package collection.
- Objects with names like SPIFFY.MSTR_, where the underscore stands for any single character, only by plans with names that begin with TRULY.

The application registration table remains as in Table 33 on page 3-51; PLANA and PACKB have total system control (but with exceptions). Table 34 shows the entries needed to register those exceptions in the object registration table.

Table 34. Table DSN_REGISTER_OBJT for system control with exceptions

QUALIFIER	NAME	TYPE	APPLMATCHREQ	APPLIDENT	APPLIDENTTYPE
KIM	VIEW1	C	Y	PLANC	P
BOB	ALIAS	C	Y	PACKD	C
FENG	TABLE2	C	N		
SPIFFY	MSTR_	C	Y	TRULY%	P

The first of those entries requires an application match for the object KIM.VIEW1: the view can be created, altered, or dropped only by the application plan PLANC. Similarly, the second entry specifies that BOB.ALIAS can be created, altered, or dropped only by the package collection PACKD. The third entry requires no application match for FENG.TABLE2: the object can be created, altered, or dropped by *any* plan or package collection. 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

It is not necessary to give a complete two-part name for every object registered in the ORT. To use incomplete names, on installation panel DSNTIPZ enter:

```
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 represented by incomplete names in the ORT need an authorizing entry in the ART.

With the new option in effect, you could add to Table 34 on page 3-53 the entries shown in Table 35.

Table 35. 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 36. That allows the applications to use sets of objects 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 was missing *** tonello *** 09/30/97 --> *.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. But 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 36 shows PLANA and PACKB registered in the ART to use sets of objects that are registered in the ORT with incomplete names.

Table 36. 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 3-52. On panel DSNTIPZ, enter:

```
CONTROL ALL APPLICATIONS ===> NO
```

2. Also on panel DSNTIPZ, enter:

   ```
   UNREGISTERED DDL DEFAULT ===> REJECT
   ```

   That option totally restricts the use of DDL statements for objects not registered in the object registration table: *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, then 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, whether or not that set has APPLMATCHREQ = Y.

Example: Table 37 on page 3-56 shows entries in the object registration table 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, say 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. We assume 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.

If we assume the installation option

```
REQUIRE FULL NAMES       ===> YES
```

then the table entries do not specify incomplete names. Hence, objects not represented in the table cannot be created in the system, except by an ID with installation SYSADM authority.

Table 37. 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 3-52. On panel DSNTIPZ, enter:

   ```
   CONTROL ALL APPLICATIONS ===> NO
   ```

2. Also on panel DSNTIPZ, enter:

   ```
   UNREGISTERED DDL DEFAULT ===> ACCEPT
   ```

   That option *does not* restrict the use of DDL statements for objects not registered in the object registration table: *any* application can use DDL for any unregistered object.

3. Register all controlled objects in the object registration table. 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 application registration table. Supply values for the APPLIDENT and APPLIDENTTYPE columns as before. You must also supply values for one additional column:

Column Name	Description
QUALIFIEROK	Use 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, was set to NO, as described in “Registering Sets of Objects” on page 3-53. Table 38 on page 3-57 shows entries in the object registration table 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 3-54. Those names cannot be interpreted as incomplete names because the objects that control them, PLANX and PACKX, are registered in Table 39 on page 3-57 with QUALIFIEROK=N.

- Two sets of objects, *.TABA and *.TABB, which are controlled by PLANA and PACKB, respectively.

Table 39 shows entries in the corresponding application registration table.

In this situation, with the combination of installation options shown above, any application can use DDL for objects *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.

Table 38. 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 39. 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

Managing the Registration Tables and Their Indexes

"Columns of the Application Registration Table (ART)" and "Columns of the Object Registration Table (ORT)" on page 3-58 describe the columns of the two registration tables.

An Overview of the Registration Tables

Columns of the Application Registration Table (ART)

APPLIDENT CHAR(18); can include pattern characters

Collection-ID of the package executing the DDL or, if there is no package, the name of the plan executing the DDL

APPLIDENTTYPE CHAR(1)

Type of application identifier (rows with other values are ignored):

C package collection name
P plan name

APPLICATIONDESC VARCHAR(30)

Optional description of the application

DEFAULTAPPL CHAR(1)

Whether all DDL should be accepted from this application:

Y Yes
N No (the default; rows without Y are ignored)

QUALIFIEROK CHAR(1)
Whether the application can supply a missing name part for objects named in the ORT, if REQUIRE FULL NAMES = NO:

Y Yes
N No (the default; rows without Y are ignored)

CREATOR CHAR(26)
Suggested use: Authorization ID that created the row

CREATETIMESTAMP TIMESTAMP
Suggested use: Time the row was created

CHANGER CHAR(26)
Suggested use: Authorization ID that last altered the row

CHANGETIMESTAMP TIMESTAMP
Suggested use: Time the row was last altered

Columns of the Object Registration Table (ORT)

QUALIFIER CHAR(8); can include pattern characters
Object name qualifier; blank, if none

NAME CHAR(18); can include pattern characters
Unqualified object name

TYPE CHAR(1)
Type of object (rows with other values are ignored):

C collection qualified name (table, view, index, synonym, or alias)
D database
T table space
S storage group

APPLMATCHREQ CHAR(1)
Whether an application naming this object must match the one named in the APPLIDENT column:

Y Yes
N No (the default; rows without Y are ignored)

APPLIDENT CHAR(18); can include pattern characters
Collection-ID of the package executing the DDL or, if there is no package, the name of the plan executing the DDL

APPLIDENTTYPE CHAR(1)
Type of application identifier (rows with other values are ignored):

C package collection name
P plan name

APPLICATIONDESC VARCHAR(30)
Optional description of the application

CREATOR CHAR(26)
Suggested use: Authorization ID that created the row

CREATETIMESTAMP TIMESTAMP
Suggested use: Time the row was created

CHANGER CHAR(26)
Suggested use: Authorization ID that last altered the row

CHANGETIMESTAMP TIMESTAMP
Suggested use: Time the row was last altered

Creating the Tables and Indexes

The application registration table (ART), the object registration table (ORT), and the required unique indexes on each of 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 those 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. Make the name of a required index from the name of its corresponding table by adding the letter I.

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

allowed in such circumstances are those that create the registration tables 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

Both tables contain columns not used by DB2. We recommend using them to audit and manage the tables as follows:

- In APPLICATIONDESC, put a more readable description of each application than the 8-character APPLIDENT column can contain.
- In CREATOR or CHANGER, put the authorization ID that created or last changed the row. There is room 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 whether or not data definition control is active, and whether or not the ART or ORT are 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 process executing

Data Sharing

The ART and ORT tables must have the same names for every member of a data sharing group.

Chapter 3-4. Controlling Access to a DB2 Subsystem

This chapter tells how to control access to the DB2 subsystem from different environments and how, while doing that, to associate a process with an intended set of authorization IDs.

External Security System: We recommend that you control access through an external security system, for which Resource Access Control Facility (RACF) is the model. “Establishing RACF Protection for DB2” on page 3-93 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 3-111. But most of the description assumes that RACF, or something like it, 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” on page 3-64. When you come to “Controlling Requests from Remote Applications” on page 3-71, you can skip everything up to “Establishing RACF Protection for DB2” on page 3-93.

Remote Requests: If you are accepting requests from remote applications, you might first want to read “Controlling Requests from Remote Applications” on page 3-71, 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. Read about those processes under “Controlling Local Requests” on page 3-64.

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 3-84. If you send requests to a remote DBMS that is not DB2 for OS/390, use the documentation for that DRDA application server.

Topics Covered in This Chapter:

Controlling Local Requests

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 3-71.)

- 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
  - Applications that connect using the Recoverable Resource Manager Services attachment facility (RRSAF). (See Section 6 of *Application Programming and SQL Guide* for more information.)
- 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.” For instructions on controlling the IDs associated with sign-on requests, see “Processing Sign-ons” on page 3-68.

IMS, CICS, RRSAF, or DDF-to-DDF connections can make 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 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 58 on page 3-65.

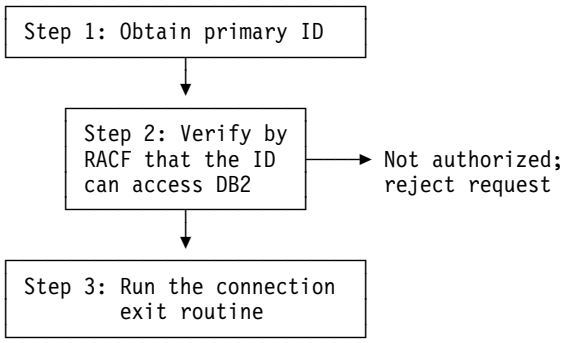


Figure 58. Connection processing

The Steps in Detail

The steps in processing connections are:

1. DB2 obtains the initial primary ID. Table 40 shows how the source of the ID depends on the type of address space from which the connection was made.

   Table 40. 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 3-71 for more details.

2. RACF is called through the system authorization facility (SAF) to check whether the ID 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 3-97. The SAF return code (RC) from the invocation determines the next step, as follows:

   If RC > **4**, then 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, then the RACF return code is checked. If that value is

   = 4, then 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 3-94.

   Not = 4, then RACF is not active. DB2 continues with the next step, but the connection request and the user are not verified.

If RC = 0, then 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."

   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 there is a value for the initial primary authorization ID, the value becomes the DB2 primary ID.
   - If there is no value (the value is blank), then the primary ID is set by default, as shown in Table 41.
   - The SQL ID is set equal to the primary ID.
   - There are no secondary IDs.

   If you *do* want to use secondary IDs, see the description in "Supplying Secondary IDs for Connection Requests." 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 41. 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 3-35, the easiest method is to use the IBM-supplied sample routine.

Pause here to 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, 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 as well as primary and SQL IDs. Installation job DSNTIJEX replaces the default connection exit routine with the sample connection exit routine; see Section 2 of *Installation Guide* for more information.

In full, the sample connection exit routine has the following effects:

- The DB2 primary ID is set in 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, as well as 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, there are no secondary IDs.
  - If RACF is active but its “list of groups” option is not active, then there is one secondary ID, the default connected group name, if that was supplied by the attachment facility.
  - 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 (but not in REVOKE status), up to a limit of 245 groups. 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 X-25.

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), then the CICS sign-on table must specify EXTSEC=YES for each signed on user that will use the sign-on exit.
- When the user signs on to a CICS terminal-owning region (TOR), the TOR must propagate the authorization ID to the CICS application-owning region (AOR). For more information on that propagation, see the description of ATTACHSEC in the applicable version of the CICS *Intercommunication Guide*.

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 using it. For instructions, see “Sample Exit Routines” on page X-26.

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. It is possible to make a new sign-on request to run the same plan without deallocating the plan and reallocating it. Nevertheless, the new sign-on can change the primary ID.

Unlike connection processing, sign-on processing does not include a check of the RACF user ID of the address space. The steps are shown in Figure 59.

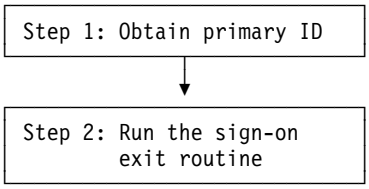


Figure 59. Sign-on processing

The Steps in Detail

The steps in processing sign-ons are:

Step 1. The initial primary ID is determined 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 not only the IMS sign-on ID but also the associated RACF connected group name, if there is one.

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 Section 2 of *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, supplied in the RCT; use AUTH=(*string*).
- The CICS group ID (eight characters); use AUTH=GROUP. That option passes to DB2 not only the CICS user ID, but also the

associated RACF connected group name. AUTH=GROUP is not a valid authorization type for transactions that do not have RACF user IDs 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 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 5 blanks); use AUTH=USER. AUTH=USER is valid only for transactions 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.
- 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 Attach Request” on page 3-75 tells 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 assumed to be the normal case; however, if an ACEE is not associated with the TCB then SIGNON uses the primary authorization id associated with the address space; that is, from the ASXB. If the new primary authorization ID was retrieved from the ACEE 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, then AUTH SIGNON also uses the value passed in the secondary authorization ID parameter to establish secondary authorization IDs. If the primary authid is not passed, but a valid ACEE is passed, then AUTH SIGNON uses the value in ACEEUSRI for the primary authorization ID if ACEEUSRL is not 0. If ACEEUSRI is used for the primary authid then 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 associated with the current RRS context using the context_key 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 which 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 Section 6 of *Application Programming and SQL Guide* .

Step 2. DB2 runs the sign-on exit routine. *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.
There are no secondary IDs.

Of course, you can replace the exit routine with one of your own devising, 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 Section 2 of *Installation Guide*.

Again, you want to distinguish carefully between two routines. The *default* sign-on routine provides no secondary IDs and has the effects described in step 2 of Processing Sign-ons. The *sample* sign-on routine provides for 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, there are no secondary IDs.
  - If RACF is active but its “list of groups” option is not active, then there is one secondary ID, 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 chapter:

- "Overview of Security Mechanisms for DRDA and SNA"
- "The Communications Database for the Server" on page 3-72
- "Controlling Inbound Connections that Use SNA Security Mechanisms" on page 3-74
- "Controlling Inbound Connections that Use TCP/IP Protocols" on page 3-81

Overview of Security Mechanisms for DRDA and SNA

SNA and DRDA have different security mechanisms. DRDA allows a user to 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 an SNA attach or using DRDA commands. DB2 for OS/390 as a requester uses SNA security mechanisms if using an SNA network connection (except for DCE) and DRDA security mechanisms for TCP/IP network connections (or when DCE authentication is chosen regardless of the network type).

Mechanisms Used by DB2 for OS/390 as a Requester

DB2 for OS/390 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 3-91
- User ID and PassTicket, described in "Sending RACF PassTickets" on page 3-92

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 3-91
- User ID and PassTicket, described in "Sending RACF PassTickets" on page 3-92

Authentication is performed based on DRDA level 3 protocols, which means that the authentication tokens are sent in DRDA security flows.

If you use a requester other than DB2 for OS/390, refer to that product's documentation.

Mechanisms Accepted by DB2 for OS/390 as a Server

DB2 for OS/390 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 as a server:

- User ID only (already verified at the requester)
- User ID and password, described in “Sending Passwords” on page 3-91
- User ID and PassTicket, described in “Sending RACF PassTickets” on page 3-92
- DCE tickets, described in “Establishing DCE Security for DB2” on page 3-106
- User ID and encrypted password, described in “Sending encrypted passwords from a workstation” on page 3-92
- 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 level 3 requester is notified that the RACF password has expired, and the requester has implemented function to allow passwords to be changed, then 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.

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.

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 level 3 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 Version 5 and subsequent releases.

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 is returned.

The Communications Database for the Server

The information under this heading, up to “Controlling Inbound Connections that Use SNA Security Mechanisms” on page 3-74, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

The communications database (CDB) is a set of DB2 catalog tables that let you control aspects of requests leaving this DB2 and requests coming in. In this

section, we concentrate on the columns of the communications database pertaining 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 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 3-92
    - User ID and RACF encrypted password (not recommended)
    - DCE security tickets, described in "Establishing DCE Security for DB2" on page 3-106
    - User ID and DRDA encrypted password, described in "Sending encrypted passwords from a workstation" on page 3-92.
    - User ID, password, and new password, described in "Allowing Users to Change Expired Passwords" on page 3-72

  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 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 encrypted.

  N
  : No, passwords are not encrypted. This is the default; any character but Y is treated as N.

When you connect to a DB2 for OS/390 partner that is at Version 5 or a subsequent release, we recommend that you 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)
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, and all are translated in the same way.

LINKNAME CHAR(8)
Identifies the VTAM or TCP/IP network locations 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 Security Mechanisms

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 attach request.

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, then 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 Section 3 of *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. We recommend that you specify partner-LU verification; it 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 Section 3 of *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 3-96.

Accepting a Remote Attach 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* attach request that comes through one of the DB2 attachment facilities—IMS, CICS, TSO, and so on. A remote attach 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 attach requests.

Conversation-level Security: Throughout this section, we assume 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 Section 3 of *Installation Guide*. That value provides more options than does “conversation” (SECACPT=CONV), which we do not recommend.

Steps, Tools, and Decisions: The steps an attach request goes through before acceptance allow much flexibility in choosing security checks. Scan Figure 60 on page 3-77 to see what is possible.

The primary tools for controlling remote attach 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 attach requests, or else you need a dummy row that allows *any* system to send attach requests (or you could have 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?” on page 3-76
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 3-77
5. “Do You Translate Inbound IDs?” on page 3-79
6. “How Do You Associate Inbound IDs with Secondary IDs?” on page 3-81

Do You Permit Access?: To permit attach 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. (There can be only one dummy row, to be used by all LUs for which there is no specific row, when making requests.) Without one of those rows, the attach 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 DCE can be used to authenticate the user. DCE 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 accepting 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 there is an O there already because you are also sending requests *to* that LU, change O to B.) Attach requests from that LU now go through sign-on processing, and its IDs are subject to translation. (For more about that, see “Do You Translate Inbound IDs?” on page 3-79.)

To manage incoming IDs through RACF, leave USERNAMES blank for that LU (or leave the O unchanged). Requests from that LU now 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 (as we described in “Verifying a Partner LU” on page 3-75). 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 3-75.) 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. To eliminate the I/O, we recommend that you 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 3-92.
- DRDA password encryption support. DB2 for OS/390 as a server supports DRDA encrypted passwords. See "Sending encrypted passwords from a workstation" on page 3-92 for more information.

If you use DCE, are users authenticated?: If your distributed environment uses DCE to manage users and perform user authentication, DB2 for OS/390 can use DCE security services to authenticate remote users. See "Establishing DCE Security for DB2" on page 3-106.

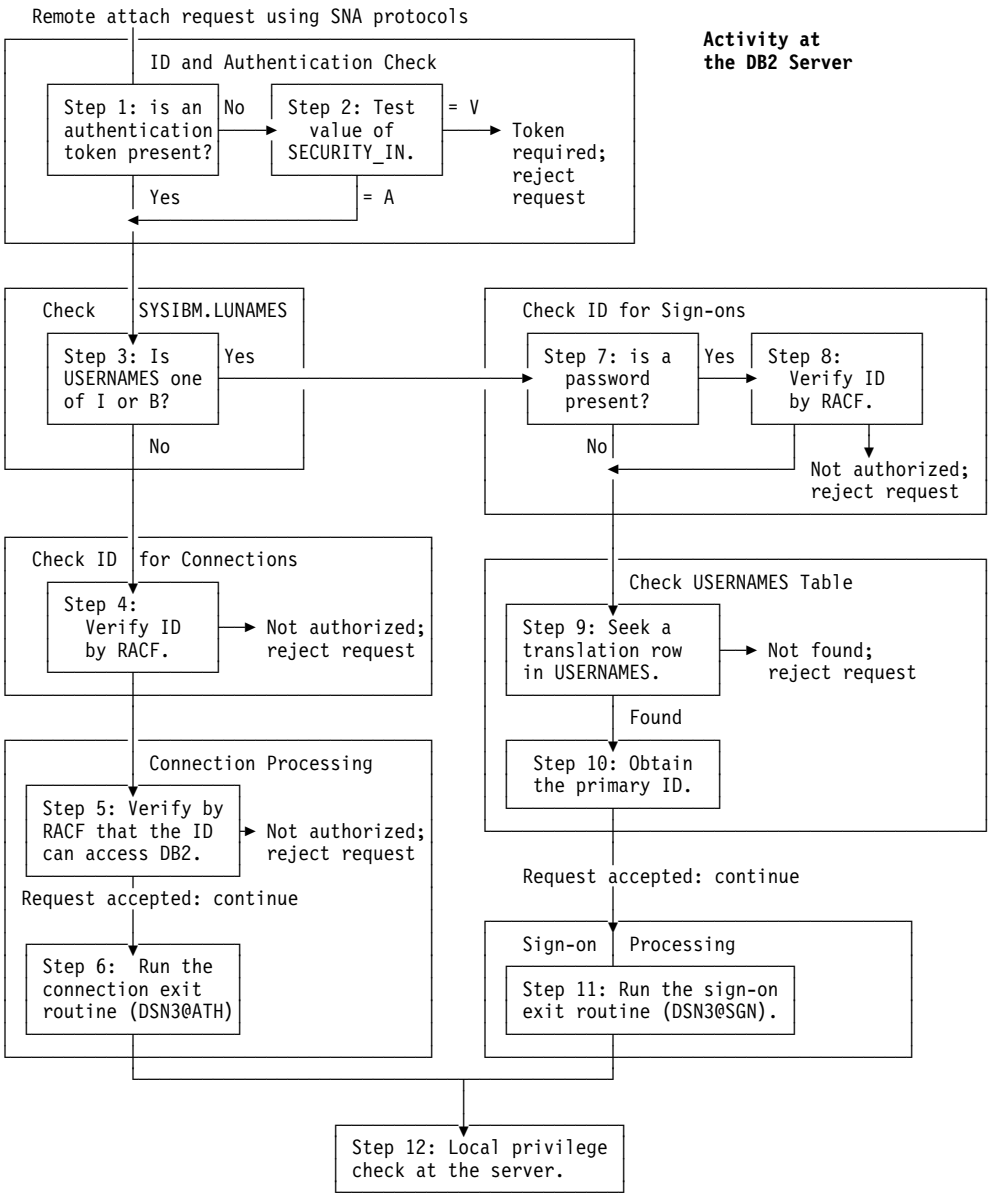


Figure 60. Steps in Accepting a Remote Attach Request from Requester that is using SNA

Details of Remote Attach Request Processing:

| Step 1. If there is no authentication token with the remote request, 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 sent from a DB2 system.

| Step 2. If the acceptance option is “verify” (SECURITY_IN = V), there must be a security token to authenticate the user. DB2 rejects the request if the token missing.

Step 3. If the USERNAMES column of SYSIBM.LUNAMES contains I or B, the authorization ID, and the plan or package owner sent by a DB2 system, 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.

Step 4. DB2 calls RACF by the RACROUTE macro with REQUEST=VERIFY to check the ID. The PASSCHK=NO option is used if there is no password, and ENCRYPT=YES is used 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:

  a. 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 attach 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 use in the verify call is the requesting LU name.

| Step 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 3-64. DB2 calls RACF by the RACROUTE macro with REQUEST=AUTH, to check whether the authorization ID is allowed to use DB2 resources 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 3-103.

Step 6. DB2 invokes the connection exit routine. The parameter list passed to the routine describes where a remote request originated.

Step 7. If there is no password, RACF is not called. The ID is checked in SYSIBM.USERNAMES.

Step 8. If there is a password, 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 the ID or password cannot be verified, DB2 rejects the request.

Step 9. DB2 searches SYSIBM.USERNAMES for a row that tells 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 there is no such row, then DB2 rejects the request.

Step 10. If an appropriate row is found, DB2 translates the ID as follows:

- If there is a nonblank value of NEWAUTHID in the row, that value becomes the primary authorization ID.
- If NEWAUTHID is blank, the primary authorization ID remains unchanged.

Step 11. The remote request is now treated like a local sign-on request; for details, see “Processing Sign-ons” on page 3-68. DB2 invokes the sign-on exit routine. The parameter list passed to the routine describes where a remote request originated. For details, see “Connection and Sign-On Routines” on page X-25.

Step 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 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 attach request comes in from DBADM1, if nothing is done to alter the ID, the wrong user can exercise privileges of DBADM1 in the local DB2 . The protection against that exposure is to translate the remote ID into a different ID before the attach request is accepted.

You must be prepared to translate the IDs of plan and package owners, as well as the primary IDs of processes that make remote requests. For the IDs sent to you by other DB2 LUs, see “What IDs You Send” on page 3-87. (Do not plan to translate all IDs in the connection exit routine: It 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 60 on page 3-77 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. We recommend *not* translating inbound IDs in an exit routine, but only through the SYSIBM.USERNAMES table.

The examples in Table 42 on page 3-80 shows what possibilities there are 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. While doing that, you can translate any incoming ID to another value. Table 43 on page 3-81 shows the search order of the SYSIBM.USERNAMES table.

Performance Considerations: In the process of accepting remote attach 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 3-75. 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 in Section 3 of *Installation Guide* .

Example: Translating Inbound IDs

Table 42. Your SYSIBM.USERNAMES Table. (Row numbers have been 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. It finds one in row 4, so the request is accepted. The value of NEWAUTHID there is blank, so ALBERT is left unchanged.
BETTY requests from LUDALLAS	DB2 searches for an entry for AUTHID=BETTY and LINKNAME=LUDALLAS; there is none. It next searches for AUTHID=BETTY and LINKNAME=blank. It finds that entry in row 5, so the request is accepted. The value of NEWAUTHID there is blank, so BETTY is left unchanged.
CHARLES requests from LUDALLAS	DB2 searches for AUTHID=CHARLES and LINKNAME=LUDALLAS; there is no such entry. It next searches for AUTHID=CHARLES and LINKNAME=blank. The search ends at row 3; the request is accepted. The value of NEWAUTHID there is CHUCK, so CHARLES is translated to CHUCK.
ALBERT requests from LUSNFRAN	DB2 searches for AUTHID=ALBERT and LINKNAME=LUSNFRAN; there is no such entry. It next searches for AUTHID=ALBERT and LINKNAME=blank; again there is no entry. Finally, it searches for AUTHID=blank and LINKNAME=LUSNFRAN; it finds that entry in row 1, so the request is accepted. The value of NEWAUTHID there 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 has been made for WILBUR, but row 1 of the SYSIBM.USERNAMES table allows any ID to 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. Once accessing DB2, WILBUR can use only the privileges 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), some IDs (ALBERT, BETTY, CHARLES) can attach from LUDALLAS, but WILBUR cannot attach if coming from LUDALLAS.

Table 43. 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) 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 attach 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. Go on now to read “Processing Connections” on page 3-64 or “Processing Sign-ons” on page 3-68.

Controlling Inbound Connections that Use TCP/IP Protocols

DRDA connections that use TCP/IP have fewer security controls than do connections using 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 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? There is no option to handle incoming IDs through DB2. All IDs must be passed on to RACF or DCE for processing.

Do You Trust the Partner? There is no concept similar to SNA's partner LU verification with TCP/IP. If your requesters support mutual authentication, use DCE 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 3-92.
- DRDA password encryption support. DB2 for OS/390 as a server supports DRDA encrypted passwords. See “Sending encrypted passwords from a workstation” on page 3-92 for more information.

If you use DCE, are users authenticated? If your distributed environment uses DCE to manage users and perform user authentication, DB2 for OS/390 can use DCE security services to authenticate remote users. See “Establishing DCE Security for DB2” on page 3-106.

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 (DSN3@ATH). TCP/IP requests do not use the sign-on exit.

Steps, Tools, and Decisions

See Figure 61 on page 3-83 for an overview of how incoming requests are handled. See “Detecting Authorization Failures (EXTENDED SECURITY)” on page 3-72 for information about security diagnostics.

1. You must first decide whether you want incoming requests to have authentication information passed along with the authorization ID: RACF passwords, RACF PassTickets, or DCE tickets.

   To indicate that you require this authentication information, specify NO on the TCP/IP ALREADY VERIFIED field of installation panel DSNTIP5 (this 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 in.

   - 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))
     ```

   - When you use DCE tickets for incoming authentication, make sure that each DCE name is registered with the DCE server and in the DCE segment of the RACF registry for the MVS that houses the DB2 server. For more information about setting up DB2 to use DCE security, “Establishing DCE Security for DB2” on page 3-106.

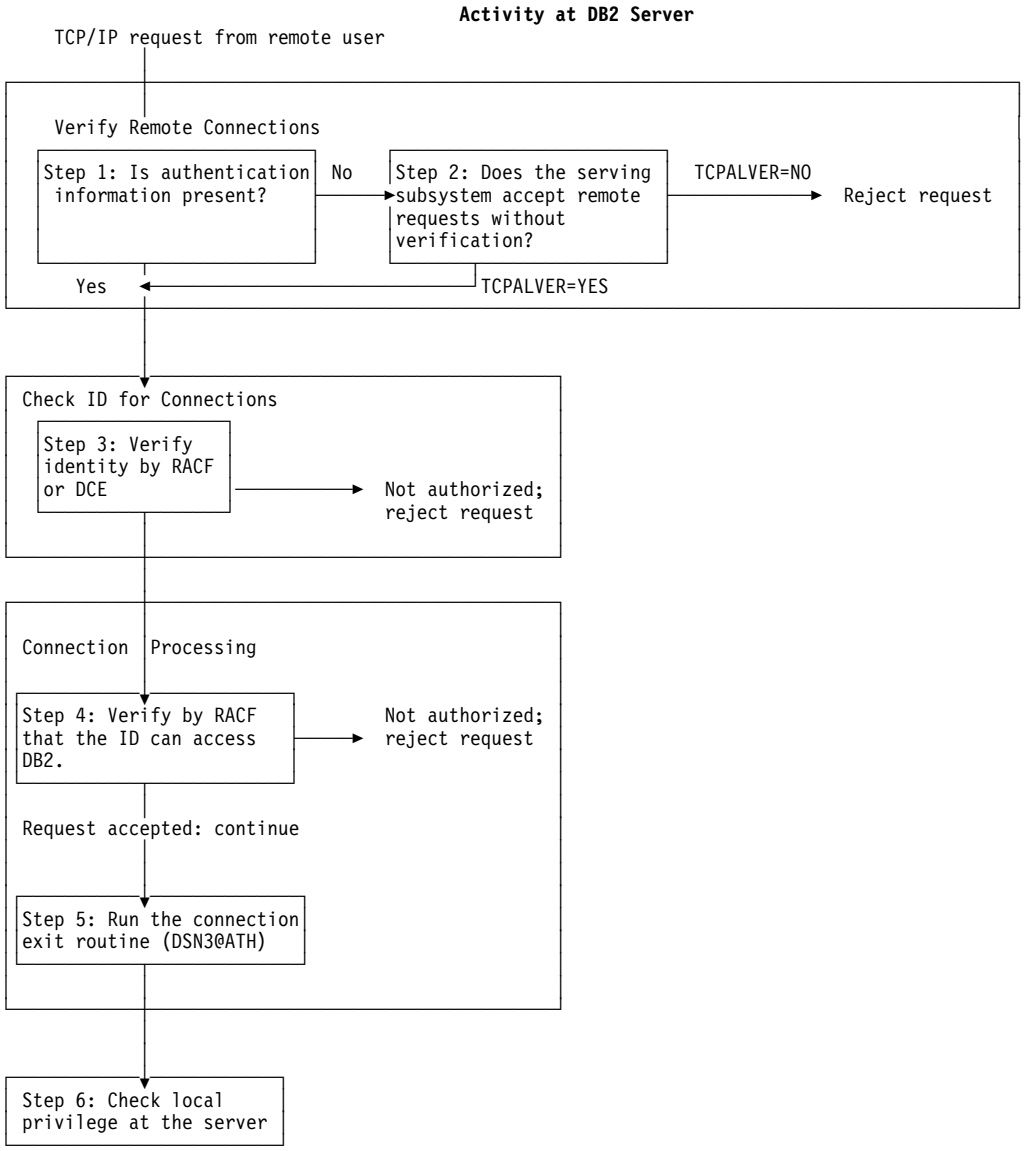


Figure 61. Steps in Accepting a Request from TCP/IP.

Details of Steps: These notes explain the steps shown in Figure 61.

Step 1. DB2 checks to see if an authentication token (RACF encrypted password, RACF PassTicket, DRDA encrypted password, or DCE ticket) accompanies the remote request.

Step 2. If there is no authentication token 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.

Step 3. The identity is a RACF ID that is authenticated by RACF if a password or PassTicket is provided, or it is a DCE principal identity that is validated by DCE, if a DCE ticket is provided. You must be sure that the ID is defined to RACF in all cases. When DCE tickets are used, the RACF ID is derived from the DCE principal identity. To do this, you must ensure that you cross-link DCE principal names with RACF IDs, as described in "Establishing DCE Security for DB2" on page 3-106.

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 been given access to the DB2 APPL. The APPL resource that is checked is the LU name that the requester used when the attach 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 use in the verify call is the string `'TCPIP'`.

If this is a request to change a password, the password is changed.

Step 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 IDs authorization against the *ssnm*.DIST resource.

Step 5. DB2 invokes the connection exit routine. The parameter list passed to the routine describes where the remote request originated.

Step 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 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, consider that the security administrator there might have chosen any of the options described in “Controlling Requests from Remote Applications” on page 3-71. You have 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 system requires.

If you are planning to send remote requests to some DBMS that is not DB2 for OS/390, you have to satisfy the requirements of that system. You probably need documentation for the particular type of system; some of the choices described in this section might not apply.

Network Protocols and Authentication Tokens: DB2 chooses how to send authentication tokens based on the network protocols being used (SNA or TCP/IP). If the request is sent using SNA, the authentication tokens are sent in the SNA attach request (FMH5), unless you are using DCE. If you use DCE, 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 under this heading, up to “What IDs You Send” on page 3-87, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

The communications database (CDB) is a set of DB2 catalog tables that let you control aspects of remote requests. In this section, we concentrate on the columns

of the communications database pertaining 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 not specifically listed elsewhere in the column.

SECURITY_OUT (CHAR 1)
: This column defines the security option that is used when local DB2 SQL applications connect to any remote server 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 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 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.

  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 used for an outbound request is the translated ID.

ENCRYPTPSWDS CHAR(1)
: This column only applies to DB2 for MVS/ESA or DB2 for OS/390 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 encrypted.

  N — No, passwords are not encrypted. This is the default; any character but Y is treated as N.

  When you connect to a DB2 for OS/390 partner that is at Version 5 or a subsequent release, we recommend that you use RACF PassTickets (SECURITY_OUT='R') instead of encrypting passwords.

USERNAMES CHAR(1)
: This column indicates whether an ID accompanying a remote attach 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.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
This column defines the DRDA security option that is used when local DB2 SQL applications connect to any remote server 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 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 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.

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)
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 are translated in the same way.

LINKNAME CHAR(8)
Identifies the VTAM or TCP/IP network locations 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.

What IDs You Send

The primary authorization ID of the process making the request is always sent to the server. This is the ID that is used for authentication at the remote server. But other IDs can accompany some requests. It's important 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 44 shows what other IDs you send for the different situations that can occur.

Table 44. 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

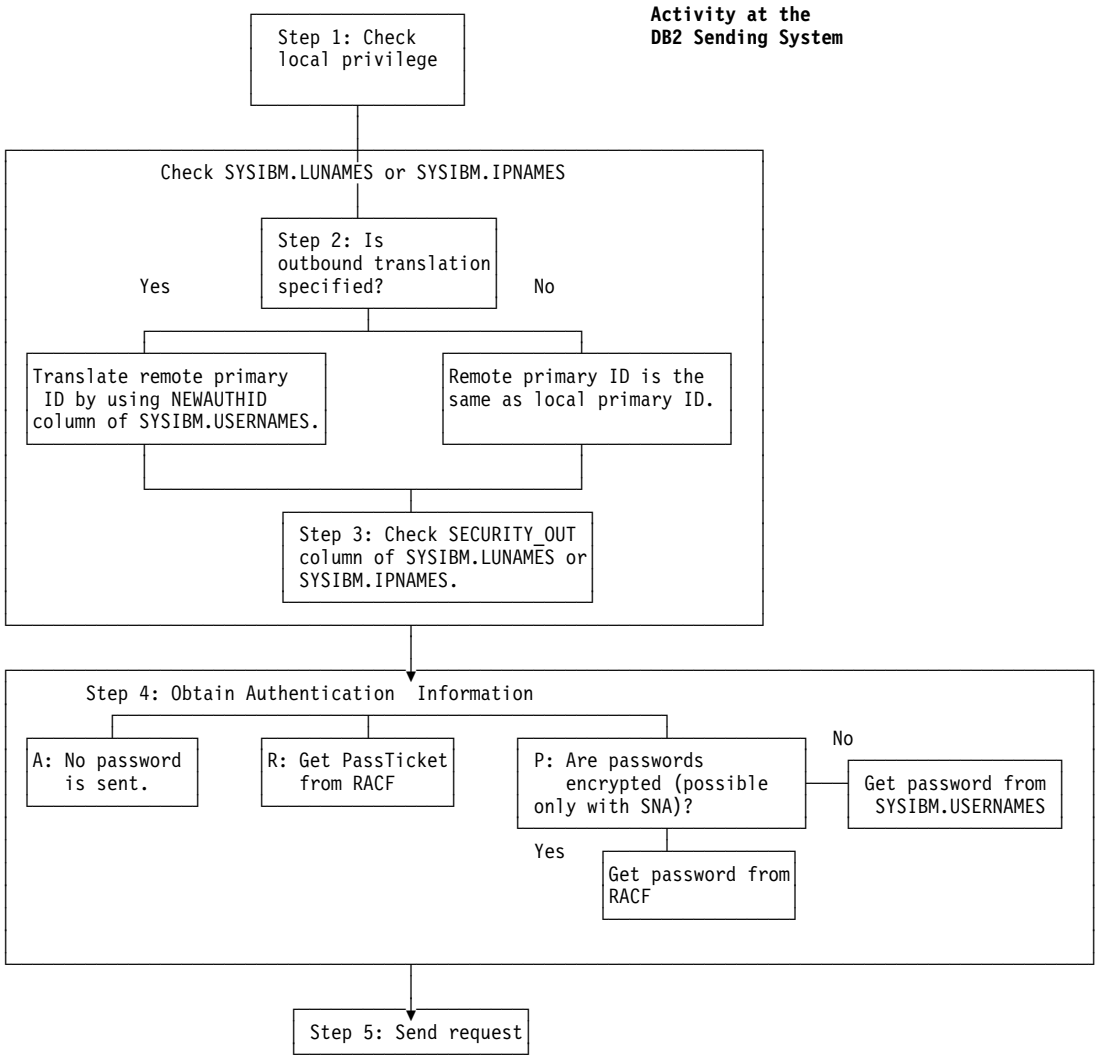


Figure 62. Steps in Sending a Request from a DB2 Subsystem

Details of Steps in Sending a Request from DB2: These notes explain the steps in Figure 62.

Step 1. The DB2 subsystem that sends the request checks whether the primary authorization ID has the privilege to execute the plan or package.

DB2 checks to see 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.)

Step 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. Be sure that these IDs are

included in SYSIBM.USERNAMES, or you will receive SQLCODE -904. DB2 translates the ID as follows:

- If there is a nonblank value of NEWAUTHID 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, then the request is rejected with a SQLCODE -904.

If column USERNAMES does not contain O or B, the IDs are not translated.

Step 3. SECURITY_OUT is checked for outbound security options. Those options are 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 3-92 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 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.

We recommend using RACF PassTickets or DCE tickets to avoid flowing unencrypted passwords over the wire. If the requester uses TCP/IP protocols, you cannot encrypt the password; therefore, the password is always obtained from RACF.

Step 4. Send the request. See Table 44 on page 3-87 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 coming *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 this in 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 as well as 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 there is no row telling 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, say LUYYY, then you generally need another set of rows to tell how your IDs are translated on outbound requests to LUYYY.

But 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 rows 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 row 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" on page 3-92
- DCE tickets, described in "Sending DCE Tickets" on page 3-92
- DRDA encrypted passwords, described in "Sending encrypted passwords from a workstation" on page 3-92

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 X-44.

DB2 for OS/390 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.

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 every 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 somewhat 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 SYSIBM.IPNAMES or SYSIBM.LUNAMES table for that system.

To set up RACF to generate PassTickets, you must 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 the remote system as it appears in the LINKNAME column of table SYSIBM.LOCATIONS. For example, the following command defines a profile for 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 DCE Tickets

DB2 for OS/390 cannot send DCE tickets for authentication. If your requester is enabled for DCE security as a requester, refer to that product's documentation for information about sending DCE tickets.

Sending encrypted passwords from a workstation

DB2 for OS/390 allows DRDA level 4 clients to flow DRDA encrypted passwords to a DB2 server. This support uses the Diffie-Hellman key distribution algorithm.[6]

To enable DB2 Connect V6 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).

[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.

Establishing RACF Protection for DB2

For purposes of illustration, suppose that the system of RACF IDs shown in Figure 63 is used to control DB2 usage.

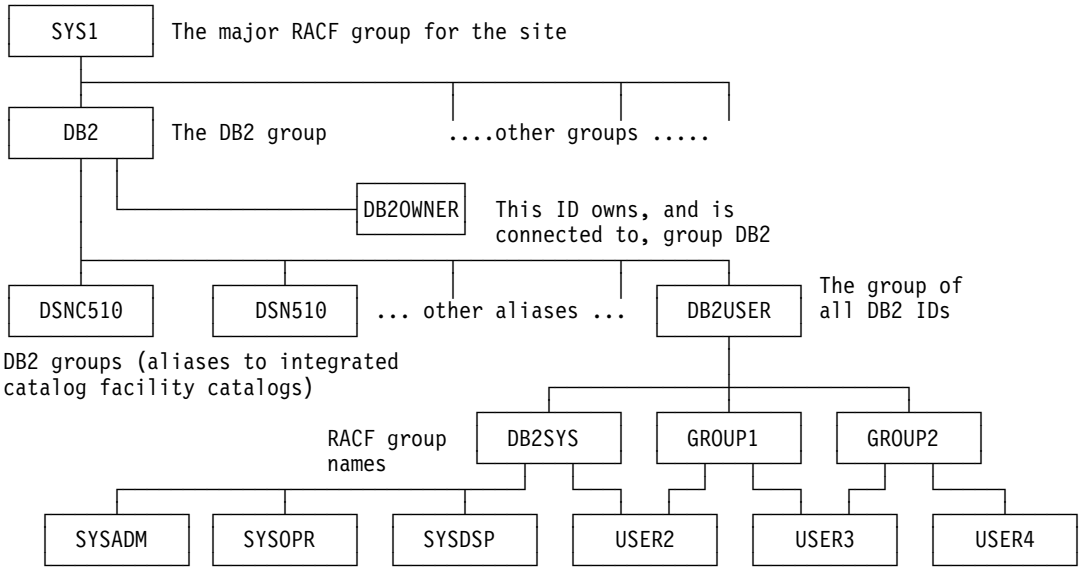


Figure 63. Sample DB2/RACF Environment

Figure 63 shows some of the relationships among the following names:

RACF ID	Use
SYS1	Major RACF group ID
DB2	DB2 group
DB2OWNER	Owner of the DB2 group
DSNC510	Group to control databases and recovery logs
DSN510	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

(There are additional RACF group names and user IDs that do not appear in the figure; they are listed in Table 45 on page 3-98.)

USER1, USER2, USER3	RACF user IDs

In order to establish RACF protection for DB2, perform the steps described below. Some are required, 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; we group them under two major objectives:

- "Defining DB2 Resources to RACF" on page 3-94 includes steps that tell RACF what to protect.
- "Permitting RACF Access" on page 3-97 includes steps that make the protected resources available to processes.

For a more thorough description of RACF facilities, see *Recource Access Control Facility (RACF) System Programmer's Guide*.

Defining DB2 Resources to RACF

To define DB2 resources to the RACF system, perform the following steps in any order:

- "Define the Names of Protected Access Profiles"
- "Add Entries to the RACF Router Table" on page 3-95
- "Enable RACF Checking for the DSNR and SERVER Classes" on page 3-96

All of the steps are required. As a result of the steps, no one can access the DB2 subsystem until RACF is further instructed 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 3-96.
- Ensure that IDs associated with stored procedures address spaces are authorized to run the appropriate attachment facility, as described in "Step 1: Control Access via the Attachment Facilities (Required)" on page 3-104.
- If you are using TCP/IP, ensure that the ID associated with the DDF address space is authorized to use OpenEdition MVS services, as described in "Establishing RACF Protection for TCP/IP" on page 3-106.

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), and TSO 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 1 to 4 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, batch, all utility jobs, DB2-established stored procedures address space, and requests 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 3-102. After you define an entry for your DB2 system 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 distributed by RACF. Figure 64 on page 3-96 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 taken when DB2 invokes the RACROUTE macro. (Refer to *Recource Access Control Facility (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, then *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, re-IPL the MVS system to cause it to use the new router table. Or 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.

Operational Note: The macro ICHRFRTB 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, those subsystems, in class DSNR with requester IDENTIFY, have their requests passed to RACF (ACTION=RACF) 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 64. 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)
```

The command must be issued by a user with the SPECIAL attribute.

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 3-104.

Enable Partner-LU Verification

With RACF 1.9, VTAM 3.3, and later releases, you can control whether two LUs using 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)
- Whether the LU pair is locked (LOCK suboperand).

For details, see *Resource Access Control Facility (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 permit processes to use the protected resources, take the following steps:

- "Define RACF User IDs for DB2 Started Tasks"
- "Add RACF Groups" on page 3-101
- "Permit Access for Users and Groups" on page 3-102

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 and the CICS address space, as well as with 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: For stored procedures, stored procedures address space entries are required in the RACF started procedures table. The associated RACF user ID and group name do not have to match those used for the DB2 address spaces, but they must be authorized to run the call attachment facility (for the DB2-established stored procedure address space) or Recoverable Resource Manager Services attachment facility (for WLM-established stored procedure address spaces).

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 65 on page 3-98 shows the sample entries for three DB2 subsystems and optional entries for CICS and IMS. (Refer to *Recource Access Control Facility (RACF) System Programmer's Guide* for a description of how code a RACF started procedures table.) The example provides for the DB2 started tasks for each of three DB2 subsystems, named DSN, DB2T, and DB2P, as well as for CICS and an IMS control region. The IDs and group names associated with the address spaces are shown in Table 45 on page 3-98.

Table 45. 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 65 (Part 1 of 4). Sample Job to Reassemble the RACF Started Procedures Table

```
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 65 (Part 2 of 4). Sample Job to Reassemble the RACF Started Procedures Table

```
*        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
*        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 65 (Part 3 of 4). Sample Job to Reassemble the RACF Started Procedures Table

```
*        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 65 (Part 4 of 4). Sample Job to Reassemble the RACF Started Procedures Table

Add RACF Groups

The details of this step depend on the groups you have defined. To add the user DB2OWNER, issue:

```
ADDUSER  DB2OWNER  CLAUTH(DSNR USER) UACC(NONE)
```

That gives DB2OWNER class authorization 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:

```
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:

```
CONNECT DB2OWNER GROUP(DB2) AUTHORITY(JOIN) UACC(NONE)
```

To make DB2 the default group for commands issued by DB2OWNER, issue:

```
ALTUSER DB2OWNER DFLTGRP(DB2)
```

To create the group DB2USER and add five users to it, issue:

```
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, it is necessary for a TSO user to log off and log on, or for a CICS or IMS user to sign on again, to pick up a new group name as a secondary authorization ID.
- A user with the SPECIAL, JOIN, or GROUP-SPECIAL RACF attribute can define new groups with any name accepted by RACF and 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 known to DB2 are usually user IDs 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, then any TSO user with the group name as a secondary ID has ownership privileges on the table. You can prevent that 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, make sure that the transaction identifier is not a RACF group; if it is, any user connected to the group has the same privileges as the transaction code.

Permit Access for Users and Groups

DB2OWNER is authorized for class DSNR, owns the profiles, and has the right to change them. The next commands allow all users belonging to 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(READE) 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 to permit the IDs for attaching systems to use the appropriate access profile. For example, to permit the IMS user ID to use the access profile for IMS on system DB2P, issue:

```
PERMIT  DB2P.MASS  CLASS(DSNR) ID(IMS)  ACCESS(READ)
```

To permit the CICS group ID to use the access profile for CICS on system DB2T, issue:

```
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 for SYSADM and one or two for SYSOPR. Those IDs can be connected to the group DB2USER, but if they are not, they certainly need to be permitted access. The next command permits the default IDs for the special authorities to use subsystem DSN through TSO:

```
PERMIT  DSN.BATCH  CLASS(DSNR) ID(SYSADM,SYSOPR) ACCESS(READ)
```

Those IDs also can be group names.

Secondary IDs: One important use of a secondary authorization ID is to identify a RACF group, to which are assigned privileges that are shared by several, perhaps many, 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 command:

```
ADDGROUP  GROUP1  OWNER(USER1) DATA('GROUP FOR DEPT. G1')
```

To permit the group to connect to the “DSN” system through TSO, DB2OWNER issues:

```
PERMIT DSN.BATCH CLASS(DSNR) ID(GROUP1) ACCESS(READ)
```

USER1 can now connect other existing IDs to the group GROUP1, using 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 in order to ensure that all threads sign on and get the correct security information.

Allowing Users to Create Data Sets: "Chapter 3-6. Auditing Concerns" on page 3-119 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  'DSNC510.DSNDBC.ST*' UACC(NONE)

PERMIT 'DSNC510.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 to access the distributed data address space (such as DSN.DIST). For example, the following RACF commands let the users represented by 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 originating 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:

```
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 your system has the RACF APPCPORT class 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 used with DB2, 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 used with DB2.

Establishing RACF Protection for Stored Procedures

This section is a summary of the procedures 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 via the Attachment Facilities (Required)"
- "Step 2: Control Access to WLM (Optional)"
- "Step 3: Control Access to Non-DB2 Resources (Optional)" on page 3-105.

Step 1: Control Access via the Attachment Facilities (Required)

The user ID 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 3-94.

The user ID 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 haven't already established a profile for controlling access from the RRS attachment as described in "Define the Names of Protected Access Profiles" on page 3-94, define *ssnm*.RRSAF in the DSNR resource class with a universal access authority of NONE.

   ```
   RDEFINE DSNR (DB2P.RRSAF  DB2T.RRSAF) UACC(NONE)
   ```

2. Activate the resource class:

   ```
   SETROPTS RACLIST(DSNR) REFRESH
   ```

3. Add user IDs that are associated with the stored procedures address spaces to the RACF Started Procedures Table:

   ```
   ⋮
            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 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, then 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, you must perform both of 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:

   ```
   RDEFINE SERVER (DB2.ssnm.applenv)
   ```

   where *applenv* is the name of the application environment associated with the stored procedure. See "Assigning Stored Procedures to WLM Application Environments" on page 5-329 for more information about application environments.

   If 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 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)
   ```

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 Y in the EXTERNAL_SECURITY column of table SYSIBM.SYSPROCEDURES for the stored procedure.

When you specify Y for EXTERNAL_SECURITY, a separate RACF environment is set up for that stored procedure. Use EXTERNAL_SECURITY=Y only when the caller must access resources outside of DB2. Figure 66 shows the user ID associated with each part of a stored procedure.

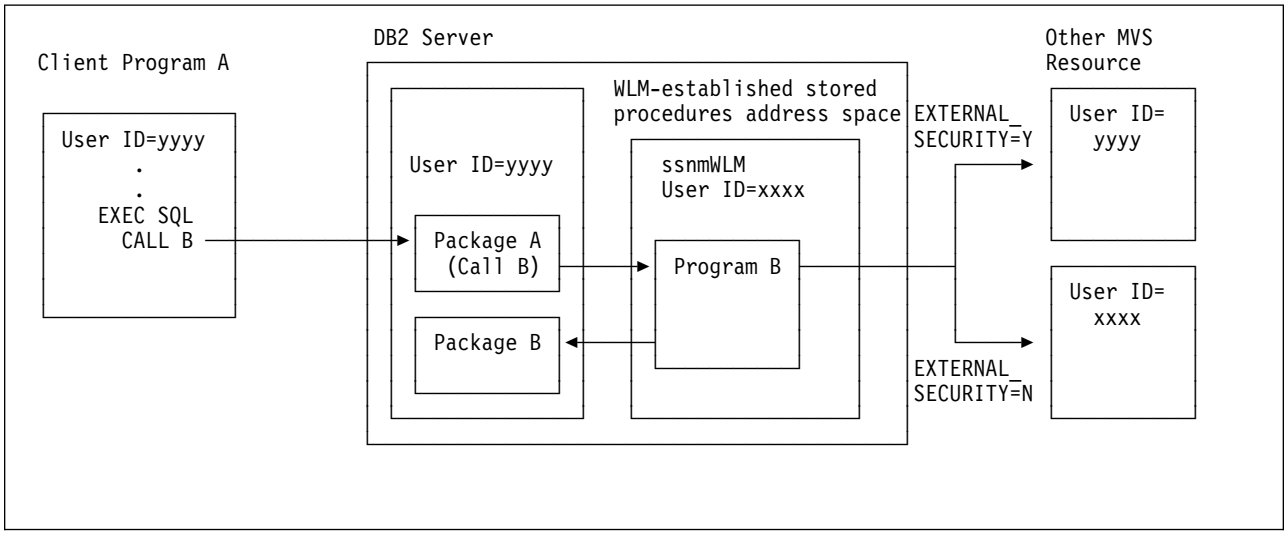


Figure 66. Accessing Non-DB2 Resources from a Stored Procedure

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.SYSPROCEDURES with EXTERNAL_SECURITY='Y'.
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 lookaside 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 associated with DB2's distributed address space must be authorized to use OpenEdition MVS services if your DB2 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:

```
ALTUSER (SYSDSP) OMVS(UID(0))
```

You must specify a UID of 0 to give root authority to the DDF address space.

Establishing DCE Security for DB2

DB2 for OS/390 can use DCE security services to authenticate remote users. With these services, remote end users can access DB2 for OS/390 by means of their DCE name and password. This same name and password is used for access throughout the network so that users do not have to maintain a separate MVS password to access DB2 for OS/390.

The DCE security technology does not require passwords to flow in readable text, and is thus secure even when used in client/server configurations. Instead, DCE

uses an authentication technology that uses encrypted *tickets* that contain authentication information for the end user.

Requirements: DB2 for OS/390 provides server support for DCE security. It requires that DB2 run on OS/390 Release 1 or subsequent releases.

To enable DB2 to use DCE authentication, do the following steps:

- "Step 1: Create a DCE Account for DB2"
- "Step 2: Define DB2 to OpenEdition Security" on page 3-109
- "Step 3: Cross-link RACF and DCE Security Information" on page 3-110
- "Step 4: Manage DB2's Server Key" on page 3-110

For more information about using DCE security, see *OS/390 OpenEdition DCE Administration Guide*.

Step 1: Create a DCE Account for DB2

To use DCE services, DB2 must have a DCE account with which it can authenticate itself to DCE. DB2's password (its DCE server key) is stored in the DCE security registry as part of its DCE account.

To create a DCE account, you must create a *principal* for the DB2 for OS/390 server. A principal is a user of the system. The DCE principal name for the DB2 server is DB2's location name, in uppercase. For our example, the location name is DB2_MVS1.

You must also create the DCE group and organization in which the account will be a member. Skip any of the steps in this procedure, as appropriate, if the principal, group, or organization have already been created.

Use the DCE dcecp administrative interface to perform the operations on the security registry. With the dcecp commands, the backslash (\) is a line continuation character. A DCE security administrator must be authorized to execute the required DCE functions. See *OS/390 OpenEdition DCE Administration Guide* for information about the authorization required for each operation.

1. Create a DCE organization in which will be a member.

   ```
   dcecp> organization create \
   > -orgid /.../sanjose.ibm.com/org1  \
   > -fullname Sample Org 1
   ```

   The remaining commands assume the same *cell*, so you do not have to explicitly include the cell name. A cell is a group of users, systems, and resources that are grouped around a common purpose and that share common DCE services.

2. Create a DCE group of which DB2 is to be a member.

   The following example uses the dcecp command to create a group named grp1:

   ```
   dcecp> group create grp1 -fullname Sample Group 1
   ```

   The command generates a Universal Unique Identifier (UUID) for the organization and displays it to you.

3. Create a DCE principal for DB2 in the Security Registry. The command generates a user ID and a UUID for the principal. Specify the principal name in uppercase.

   ```
   dcecp> principal create DB2_MVS1 -fullname Sample DB2 Server
   ```

4. Add DB2 to its DCE group:

   ```
   dcecp> group add grp1 -member DB2_MVS1
   ```

5. Add DB2 to its DCE organization:

   ```
   dcecp> organization add org1 -member DB2_MVS1
   ```

6. Create a server account for DB2:

   You must specify your DCE password (xxxxx in the example that follows) to execute this command. Part of the account information is the server's key, entered in the password attribute. In the following example, the key is represented by the string “temp.db2.key.”

   ```
   dcecp> account create DB2_MVS1 -group grp1 -org org1 \
   > -mypwd xxxxx -password temp.db2.key    \
   > -acctvalid no -client no \
   > -home / -server yes
   ```

7. Add DB2's server key to the default key table. A key is like a password for noninteractive principals, like DB2. Its key is stored in a keytab file.

   a. Set the server key in the keytab file to the current password for the registry:

      ```
      dcecp> keytab add /.:/hosts/hostname1/config/keytab/self \
      > -member DB2_MVS1 -key temp.db2.key  -version 1
      ```

      The version must match the current password version in the registry.

   b. Specify -registry and -random, to generate a random key for DB2 and to update both the DCE keytab file and the registry with this key.

      ```
      dcecp> keytab add /.:/hosts/hostname1/config/keytab/self -member \
      > DB2_MVS1 -random -registry
      ```

      where hostname1 is the DCE hostname of the MVS system.

Data Sharing Considerations: All members of a data sharing group must belong to the same DCE cell. Each member of the data sharing group must use the same principal name, the location name in upper case. Each member has its own local keytab file, but all keytab files must share the same key.

1. Perform tasks 1 through 5 in “Step 1: Create a DCE Account for DB2” on page 3-107 You can perform these tasks from any host in the cell.

2. Create a server account for DB2.

   You must specify your DCE password (xxxxx in the following example) to execute this command. Part of the account information is DB2's server key, entered in the password attribute. In the following example, the key is represented by the string “real.db2.key.”

   ```
   dcecp> account create DB2_MVS1 -group grp1 -org org1 \
   > -mypwd xxxxx -password real.db2.key    \
   > -acctvalid no -client no \
   > -home / -server yes
   ```

3. On each member of the data sharing group, add DB2's server key to the default local keytab file:

```
dcecp> keytab add /.:/hosts/hostname1/config/keytab/self \
> -member DB2_MVS1 -key real.db2.key  -version 1
```

The key in the keytab must match the password for the server account in the registry, and each member of the Sysplex must share the same key.

Step 2: Define DB2 to OpenEdition Security

Define DB2 to OpenEdition security by using RACF, or an equivalent external security subsystem.

The distributed data facility is a started-task address space named *ssnm*DIST, where *ssnm* is the subsystem name. In the following example, the *ssnm.*DIST address space has the RACF user ID SYSDSP in the RACF group DB2SYS.

Define SYSDSP as an OpenEdition MVS (OMVS) user as follows:

1. Using the RACF ALTGROUP command, create an OMVS segment in the RACF group profile:

   ```
   ALTGROUP DB2SYS OMVS (GID(n))
   ```

   There is no correlation between the RACF group and the DCE group. Choose a value *n* for the RACF group. The value can be the same as the group ID generated for the DCE group, but it need not be. RACF does not require unique group IDs.

2. Create an OMVS segment in the RACF user profile:

   ```
   ALTUSER  SYSDSP  OMVS(UID(0) HOME('/u/sysdsp') PROGRAM('/bin/sh'))
   ```

   There is no correlation between the RACF user ID and the DCE user ID (UID). The command above puts the home directory for SYSDSP in the user subdirectory named /u/sysdsp.

   You must specify a UID of 0 to give root authority to the DDF address space, which enables DDF to access the DCE default keytab, /krb5/v5srvtab.

3. Use the OMVS command mkdir to create the home directory (/u/sysdsp) to DDF.

4. Permit the SYSDSP user ID to access the RACF identity mapping function. The IRR.RDCERUID facility must already have been defined when RACF's DCE support was installed.

   ```
   PERMIT IRR.RDCERUID CLASS(FACILITY) ID(SYSDSP)
   ```

5. To allow DCE messages related to DCE processing to be displayed at the console, use the OMVS OEDIT command to create an default environment variable (envar) file in the user home directory (/u/sysdsp, in our example) . The envar file must contain the following statement:

   ```
   _EUV_SVC_MSG_LOGGING=CONSOLE_LOGGING
   ```

   See *OS/390 OpenEdition DCE Administration Guide* for more information about the envar file.

Data Sharing Considerations: All members in the Sysplex can share the same RACF database, in which case no additional steps are required in a Sysplex. However, if the RACF database is not shared, then you must define DB2 as an OMVS user to each RACF database.

Step 3: Cross-link RACF and DCE Security Information

In this step, you make sure that RACF users are known to DCE or vice versa. This section describes at a high level how you use the DCE MVSIMPT and MVSEXPT utilities to cross-link principals defined in the DCE registry with user IDs in the RACF database. For more information, see *OS/390 OpenEdition DCE Administration Guide*.

The following example assumes that you are defining an existing set of RACF users to DCE and shows you how to import those users to DCE. DCE MVSIMPT and MVSEXPT utilities also let you export DCE users to RACF. To cross-link RACF and DCE users, perform the following steps:

1. In RACF, create a DCE segment for each RACF user who will be cross-linked to the DCE registry.
2. Run the RACF database unload utility to create a file containing these selected users.
3. Sort the unloaded file.
4. Copy the file to a hierarchical file system (HFS) file.
5. Log onto DCE.
6. If needed, edit the HFS file containing the RACF information.
7. Run the MVSIMPT utility, and specify Pass1 to create a file that contains dcecp command options with which to populate the DCE registry. Repeat this step as often as necessary to refine the output file. You can also edit the file to override or correct automatically generated options and values.
8. Run the MVSIMPT utility, and specify Pass2 to invoke dcecp with the options that were generated during Pass1.

Data Sharing Considerations: If the RACF database is not shared, you must cross-link user information on each member of the data sharing group.

Step 4: Manage DB2's Server Key

To ensure the integrity of DB2's server key, change the key periodically. Each server key has a version number. Unexpired tickets that are created using the old server key are honored because they refer to a specific version of the server key.

To change a key, add a new key to the default keytab:

```
dcecp> keytab add /.:/hosts/hostname1/config/keytab/self
> -member DB2_MVS1 -random -registry
```

where hostname1 is the DCE hostname of the MVS system.

If the DB2 server key is compromised, you must remove the server key, which invalidates all tickets created using that server key. To remove a server key from the keytab, enter the following decdp command:

```
dcecp> keytab remove /.:/hosts/hostname1/config/keytab/self \
> -member DB2_MVS1
```

Data Sharing Considerations: To change the server key for DB2, perform the following steps:

1. Add the new key to the default keytab on all except one member of the Sysplex. Specify a version number *n* to distinguish the new key from old keys. This operation updates only the local keytab.

   ```
   dcecp> keytab add /.:/hosts/hostname1/config/keytab/self  \
   > -member DB2_MVS1 -key new.db2.key  -version n
   ```

2. On the excluded system, enter the following command to update the registry and this DB2 member's local keytab:

   ```
   dcecp> keytab add /.:/hosts/hostname1/config/keytab/self \
   > -member DB2_MVS1 -key new.db2.key  -version n  \
   > -registry
   ```

   Tickets for DB2 are now generated using the new server key.

To invalidate a server key that might have been compromised, you must remove the server key from each member system.

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 3-5. Protecting Data Sets

To protect fully 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. We recommend using RACF, or a similar external security system, to control access to the data sets just as it controls access to the DB2 system. “Controlling Data Sets through RACF,” below, tells how to create RACF profiles for data sets and allow their use through DB2.

Alternatively, you can protect data sets by VSAM or MVS passwords. “Protecting Data Sets by Passwords” on page 3-116 tells how. Neither operating system nor VSAM passwords are supported, however, for data sets controlled by the Storage Management Subsystem (SMS). To protect those, you must use RACF or a similar external security system.

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 3-94, and shown in Figure 63 on page 3-93. 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 DSNC510; for distribution, target, SMP, and other installation data sets, it is DSN510. DB2OWNER can create groups that control those data sets, by issuing:

```
ADDGROUP DSNC510 SUPGROUP(DB2) OWNER(DB2OWNER)
ADDGROUP DSN510 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 3-97, we chose SYSDSP as 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:

  ```
  ADDSD  'DSNC510.LOGCOPY*' UACC(NONE)
  PERMIT 'DSNC510.LOGCOPY*' ID(SYSDSP) ACCESS(ALTER)
  ```

- For archive logs:

  ```
  ADDSD  'DSNC510.ARCHLOG*' UACC(NONE)
  PERMIT 'DSNC510.ARCHLOG*' ID(SYSDSP) ACCESS(ALTER)
  ```

- For bootstrap data sets:

```
ADDSD  'DSNC510.BSDS*'    UACC(NONE)
PERMIT 'DSNC510.BSDS*'    ID(SYSDSP) ACCESS(ALTER)
```

- For table spaces and index spaces:

```
ADDSD  'DSNC510.DSNDBC.*' UACC(NONE)
PERMIT 'DSNC510.DSNDBC.*' ID(SYSDSP) ACCESS(ALTER)
```

- For installation libraries (started tasks do not need control):

```
ADDSD 'DSN510.*'          UACC(READ)
```

- For other general data sets:

```
ADDSD  'DSNC510.*'        UACC(NONE)
PERMIT 'DSNC510.*'        ID(SYSDSP) ACCESS(ALTER)
```

Although all of those are not absolutely necessary ('DSNC510.*' includes them all), the sample shows how generic profiles can be created for different types of data sets. Some parameters, such as universal access, could vary among the types. In the example, installation data sets (DSN510.*) 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 and every archive log created, which means you cannot use generic profiles for these data sets.

To protect VSAM data sets, use the cluster name. The data component names need not be protected, because the cluster name is used for RACF checking.

Access by Stand-Alone DB2 Utilities: The DB2 utilities DSN1COPY and DSN1PRNT access table space and index space data sets outside DB2 control. DSN1LOGP accesses the recovery log data sets (active logs, archive logs, and the bootstrap data sets) outside DB2 control. DSN1CHKR accesses DB2 directory and catalog table spaces outside DB2 control. The Change Log Inventory and Print Log Map utilities access the bootstrap data sets. Those are batch jobs, protected by the USER and PASSWORD options on the JOB statement. To provide a value for the USER option, say SVCAID, issue the following statements:

- For DSN1COPY:

```
PERMIT 'DSNC510.*' ID(SVCAID) ACCESS(CONTROL)
```

- For DSN1PRNT:

```
PERMIT 'DSNC510.*' ID(SVCAID) ACCESS(READ)
```

- For DSN1LOGP:

```
PERMIT 'DSNC510.LOGCOPY*' ID(SVCAID) ACCESS(READ)
PERMIT 'DSNC510.ARCHLOG*' ID(SVCAID) ACCESS(READ)
PERMIT 'DSNC510.BSDS*'    ID(SVCAID) ACCESS(READ)
```

- For DSN1CHKR:

```
PERMIT 'DSNC510.DSNDBDC.*' ID(SVCAID) ACCESS(READ)
```

- For Change Log Inventory:

```
PERMIT 'DSNC510.BSDS*' ID(SVCAID) ACCESS(CONTROL)
```

- For Print Log Map:

```
PERMIT 'DSNC510.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.

With RACF, you can permit programs, rather than user IDs, to access objects. Thus, IDs that are not authorized to access the log data sets can perhaps do so by running the DSN1LOGP utility. Access to database data sets can be permitted 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 3-14.) The following command adds the two default IDs that have those 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(DSNC510) AUTHORITY(CREATE) UACC(NONE)
CONNECT (SYSADM)         GROUP(DSN510)  AUTHORITY(CREATE) UACC(NONE)
```

The next set of commands give the IDs complete control over DSNC510 data sets. The system administrator IDs also have complete control over the installation libraries. The system programmer IDs should also be given that control.

```
PERMIT 'DSNC510.LOGCOPY*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC510.ARCHLOG*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC510.BSDS*'    ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC510.DSNDBC.*' ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSNC510.*'        ID(SYSADM SYSOPR) ACCESS(ALTER)
PERMIT 'DSN510.*'         ID(SYSADM)        ACCESS(ALTER)
```

Allowing DB2 Authorization IDs to Create Data Sets

The next command connects several IDs, already connected to the DB2USER group, to group DSNC510 with CREATE authority:

```
CONNECT (USER1 USER2 USER3 USER4 USER5)
         GROUP(DSNC510) AUTHORITY(CREATE) UACC(NONE)
```

Those IDs can now explicitly create data sets whose names have DSNC510 as the high-level qualifier. Any such data sets created by DB2 or these RACF user IDs are protected by RACF. Other RACF user IDs are prevented by RACF from creating such data sets.

If no option was supplied for PASSWORD on the ADDUSER command that added those IDs, the first password for the new IDs is the name of the default group,

DB2USER. The first time that IDs sign on, they all use that password, but must change it during their first session.

Protecting Data Sets by Passwords

As an alternative to defining data sets as RACF resources, you can use VSAM passwords, as described below. For non-VSAM data sets you can use MVS passwords. However, you cannot use MVS or VSAM passwords for data sets controlled by the storage management subsystem (SMS).

Attention: Protecting data sets by passwords, rather than by a security subsystem, is not recommended. A security subsystem provides far greater protection. Also, it is possible that DB2 support for passwords might be removed in some later release.

VSAM Passwords

You can use VSAM passwords for the bootstrap data sets, active log data sets, directory and catalog data sets, and data sets for user databases. To use the data set, the user must supply the password. For example, in creating a table space, you supply the password with the DSETPASS clause of CREATE TABLESPACE.

The password does not apply to data sets managed by SMS; data sets defined to SMS should be protected by RACF or some similar external security system.

Bootstrap Data Sets

The bootstrap data sets (BSDSs) can contain passwords for the active and archive log data sets and the system catalog. Therefore, they also must be password-protected.

The BSDSs are allocated to DB2 and to the print log map and change log inventory utilities, when they are running. The BSDSs can also be allocated to a stand-alone utility, like DSN1LOGP. It is MVS that prompts for and verifies the BSDS password; the process is not apparent to DB2.

Active Log Data Sets

Information about an active log data set, including its password, is put in the BSDSs by the change log inventory utility.

DB2 provides the password for an active log data set when the data set is opened, bypassing normal MVS operator intervention. If a utility job contains a DD statement for the BSDS, the utility uses the passwords contained in the BSDS Otherwise, if there is a DD statement for the log, it relies on password prompting when the log data set is opened.

For information about the bootstrap data set and the log data sets, see “Chapter 4-3. Managing the Log and the Bootstrap Data Set” on page 4-83.

Directory and Catalog Data Sets

At installation time, the DSNTIPP installation panel gives you the option of overriding the default catalog and directory password. The resulting password, whether defined by you or the default, is then stored in the BSDS as the password for each catalog and directory table space.

When the catalog and directory data sets are installed, the passwords for each of the data sets can be changed individually by using the SYSTEMDB statement of the change log inventory utility.

Database Data Sets

When you create a data set to be used by a DB2 database, you can include the VSAM master password (MASTERPW). You can establish a VSAM password for such a data set by one of these methods:

- Define the data set and set the VSAM password. Then, when you issue a CREATE TABLESPACE or CREATE INDEX statement, give the VSAM password again in the DSETPASS clause.
- Issue CREATE TABLESPACE or CREATE INDEX without defining the data set. Then DB2 defines the data set automatically, and you can establish a VSAM password at that point. With this method, you need to give the password only once.

All data sets associated with the same table space or index must have the same password. The password is stored in the DB2 subsystem catalog and in the integrated catalog facility catalog that lists the data sets. You can change it with ALTER TABLESPACE or ALTER INDEX, but the change is not effective until you use access method services to change it in the integrated catalog facility catalog entry.

Integrated Catalog Facility Catalog

If your integrated catalog facility catalog is password protected, you must supply a password when you create a storage group. That is the control or master password of the integrated catalog facility catalog that lists all dynamically created data sets in the storage group. For an example, see “CREATE STOGROUP Statement” on page 2-83.

The integrated catalog facility passwords are used by DB2 in access method services DEFINE, ALTER, and DELETE commands. They are stored in the SYSIBM.SYSSTOGROUP table of the DB2 subsystem catalog.

MVS Passwords

You can establish MVS passwords for archive log data sets and DB2 libraries.

Archive Log Data Sets

You can give a single password for all archive log data sets when DB2 is installed by using the DSNTIPP installation panel. If you do, a dynamically-created archive log data set is given the directory password, and the data set name and password are put into the MVS password data set. DB2 uses the password and stores it in the BSDS.

The password for the archive log data sets can be changed by using the ARCHIVE statement in the change log inventory utility. All of the archive log data sets created

after the password is changed use the new archive log data set password. Password protection for the archive log data sets can also be removed using NOPASSWD in the ARCHIVE statement of the change log inventory utility.

DB2 Libraries

You can give MVS passwords for all DB2 target and distribution libraries immediately after they are created. For information about establishing MVS passwords, refer to the appropriate MVS publication. For information about using the TSO PROTECT command to create passwords, see *TSO/E Command Reference*.

Chapter 3-6. Auditing Concerns

This chapter provides answers to some fundamental auditing concerns. 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 system. 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 not only the name of the object and the ID that received the privilege, but 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 3-44.

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 made to the structure of data (as in dropping a table) or data values (as in 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 DCE IDs to RACF IDs, and other activities of interest to auditors.

Other auditing concerns are dealt with later in this chapter; see:

How Can I Tell Who Has Accessed the Data?

The information under this heading, up to "Other Sources of Audit Information" on page 3-125 is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page xi.

The DB2 audit trace can tell who has accessed data. Briefly, the audit trace, when started, 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 an option of the CREATE and ALTER statements.

Options of the Audit Trace

You control most of what follows by the START TRACE and STOP TRACE commands.

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. It might happen that the translated ID at the requesting site is unknown to the server. In that case, the primary ID cannot be used to gather all audit records for a user that accesses remote data. The AUTHCHG record also shows the values of all secondary authorization IDs 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, then there are no problems of individual accountability. But 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

Not everything is recorded in the audit trace. The actual data changed 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.

Not all utilities are 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 is not audited.

All of that is consistent with the aim of providing a moderate volume of data with a low impact on performance. Even so, in choosing classes of events to audit, consider that you might ask for more data than you care to process.

Audit Class Descriptions

You choose events to audit by giving one or more numbers, to identify classes of events, when you start the trace. 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:

Audit Class Events Traced

1 Access attempts denied by DB2 for inadequate authorization. This class is the default, and can be started by choosing “1” or by choosing no class.

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, there are correspondingly many audit records.) 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, and could only prevent the deletion from the parent table. This is also true when the rule is CASCADE or SET NULL and could change the dependent table.

5 All read accesses to tables 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 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 DCE ID.

8 The start of a utility job, and the end of each phase of the utility.

9 Various types of records 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 X-177. It can be useful to have audit traces going at all times for IDs with SYSADM authority, for instance, because they have complete access to every table. If you have a network of DB2 systems, tracing multiple authorization IDs might be necessary 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 * 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 going 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 started by the last example:

```
-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 there are two audit traces 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 started by the last example 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. To choose to audit a table, use the AUDIT clause in the CREATE TABLE or an 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 DSN8510.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 DSN8D51A.DSN8S51D
  AUDIT CHANGES;
```

That example changes the one under “Department Table (DSN8510.DEPT)” on page X-8 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:

```
ALTER TABLE DSN8510.DEPT
  AUDIT ALL;
```

The statement is effective whether or not the table had been chosen for auditing before.

To prevent all auditing of the table, issue:

```
ALTER TABLE DSN8510.DEPT
  AUDIT NONE;
```

For CREATE TABLE, the default audit option is NONE. For ALTER TABLE, there is no default; 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 4 or 5. Use audit class 3 to tell whether there has been an interval when auditing was turned off for some table.

If ALTER TABLE turns auditing on or off for a specific table, then plans and packages that use the table are invalidated and must be rebound. Changing the auditing status does not affect plans, packages, nor dynamic SQL statements that are currently running. The change is effective only for plans, packages, or dynamic SQL statements that begin running after ALTER TABLE has completed.

You do not create catalog tables and cannot alter them; hence, you cannot audit the catalog tables.

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 X-182 and “Recording GTF Trace Data” on page X-183 for information. The records are of SMF type 102, as are performance trace records.

All of the 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 X-107. The IFCIDs for each trace class are listed with the description of the START TRACE command in Chapter 2 of *Command Reference*.

If you send trace records to SMF (the default), it is possible to lose data in the circumstances described below.

- SMF fails while DB2 continues running.
- An unexpected abend (like a TSO interrupt) occurs while DB2 is transferring records to SMF.

In those circumstances, SMF records the number of records lost. MVS has an option to stop the system rather than to lose SMF data.

Reporting the Records

Among other things, the audit trace records can tell you:

- 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 might:

- Use DB2PM. See “DB2 Performance Monitor (DB2 PM)” on page X-184 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.

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

  Tables containing sensitive data, like employee salary records, should probably be defined with AUDIT ALL. 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 also want to record any ALTER or DROP operations that affect the data. Use audit classes 3, 4, and 5.

- Grants of critical privileges

  Authorities, like SYSADM and DBADM, and explicit privileges over sensitive data, like an update privilege on records of accounts payable, must be monitored carefully. 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

  Some of those are only user errors, but others can be attempts to violate security. All must be investigated. If you have sensitive data, always use trace audit class 1. You might report by table or by authorization ID.[7]

Other Sources of Audit Information

As well as the audit trace, there are DB2 traces for other purposes also. The accounting, statistics, and performance traces might be of interest. Read about them under “DB2 Trace” on page X-177. DB2PM is useful to print reports of those traces, too; see “DB2 Performance Monitor (DB2 PM)” on page X-184.

The recovery log, though it is not an all-purpose log, can be useful for auditing. Information from the log can be printed using the DSN1LOGP utility. For example, the summary report can show what table spaces were updated within the range of the log that was scanned. The REPORT utility can tell you what log information is available and where it is located. For information on running DSN1LOGP and REPORT, see *Utility Guide and Reference*.

Image copies of table spaces are generated in typical recovery procedures for use by the RECOVER utility. You can inspect those copies, or use them to recover a table space to a particular point in time, to help narrow the time period in which some change was made. For guidance in using COPY and RECOVER, see “Chapter 4-6. Backing Up and Recovering Databases” on page 4-123.

The MVS console log contains messages about exceptional conditions encountered during DB2 operation. Inspect it for symptoms of problems.

[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.

What Security Measures Are in Force?

As an auditor, you are interested in the privileges and authorities held by IDs in the DB2 system. Read “Chapter 3-2. Controlling Access to DB2 Objects” on page 3-13.

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 Section 2 of *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 3-44.

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 is being used. Similar reports can tell you what IDs are privileged to use DB2 data sets and other resources. For instructions on obtaining such reports, you need the documentation from the external security system; for example, *Recource Access Control Facility (RACF) System Programmer's Guide*.

Another security measure, data definition control, 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 3-3. Controlling Access Through a Closed Application” on page 3-49 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, look at the complete installation panel in Section 2 of *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 headings below relate the operations to typical auditing concerns.

Is Required Data Present? Is It of the Required Type?

The pertinent control on the presence of data is to define columns with the NOT NULL attribute.

The assignment of column data types and lengths 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 “Column Specifications” on page 2-41.

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. For suggestions about indexes, see “Chapter 2-5. Designing Indexes” on page 2-51.

Has Data a Required Pattern? Is It in a Specific Range?

Table check constraints enhance the ability to control data integrity. 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 6 of *SQL Reference*.

General-use Programming Interface

An alternate 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 satisfying the WHERE clause can be entered through V1. For more information on creating and using views, see “Implementing Your Views” on page 2-105.

End of General-use Programming Interface

However, a view cannot be used with the LOAD utility, but that restriction does not apply to user-written exit routines. There are several types of user-written routines that 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 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 intended for a single column, and apply only to short string columns. But 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 X-25 for information about using exit routines.

Is New Data in a Specific Set? Is It Consistent with Other Tables?

This is exactly the question of referential integrity, a key feature of DB2. When primary and foreign keys are defined, 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 “Chapter 2-3. Maintaining Data Integrity” on page 2-19.

The method can be used 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 probably appropriate.

DB2 does not enforce referential constraints across subsystems.

What Ensures That Concurrent Users Access Consistent Data?

If you do not use Uncommitted Read (UR) isolation, the control is done by locks and is automatic. There is much you can do to trade locking resources among concurrent users, but there is nothing you can do to violate the basic principle of locking control. No program can access data that has been changed by another program but not yet committed.

However, if you use Uncommitted Read (UR) isolation, you can violate that principle. Uncommitted read (UR) isolation allows 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, run with performance trace class 3 on.

Consistency Between Systems: Where an application program writes data to both DB2 and IMS, or DB2 and CICS, the subsystems prevent concurrent use 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 4-109.

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 it asks, for each set of data, whether the opening balance and the control totals plus transactions processed 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. Where the check is wanted, it must be designed into the application program. For example, the program can maintain a table containing control information that balances update transaction control totals and field balances against a user's view. The table contains the view name, authorization ID, number of logical rows in the view (not the same as the number of physical rows in the table), numbers of insert and update transactions, opening balances, totals of insert and update transaction amounts, and any relevant audit trail information like the date, time, terminal ID, job name, and so on. 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, it writes a report that verifies control total and balancing information.

Summary

The set of techniques suggested to answer the foregoing questions is not exhaustive. Other combinations 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.

How Can I Tell That Data is Consistent?

It is not enough to control data entry; the results must also be verified. 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 Section 3 of *Utility Guide and Reference* .

General-use Programming Interface

SQL Queries

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 created on page 3-127, 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 “Chapter 2-11. Using the Catalog in Database Design” on page 2-117.

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, checks are made to verify that the modifications do not produce inconsistent data.

CHECK Utility

The CHECK utility is also pertinent.

- 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?
- 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: Is each value in a row within the range specified for that column when the table was created?

See *Utility Guide and Reference* for 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 what table spaces are in that status by using the DISPLAY DATABASE command with the RESTRICT option. See Chapter 2 of *Command Reference* for information about using this command.

REPORT Utility

You might want to determine what table spaces contain a set of tables that are interconnected by referential constraints. For that you can use the REPORT utility, as described in “Implications for COPY, QUIESCE, RECOVER, and REPORT” on page 2-34.

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 system or its databases) and any abend of DB2. The information recorded should include: command or condition type, date, time, authorization ID of the person issuing the command, and database condition code.

The information can be obtained 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. It should be reported daily and kept in a history file for comparison. Where 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 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 SQLCODE or SQLSTATE for the return code issued after an SQL statement has been run. DB2 records, on SMF, the occurrence (but not the cause) of physical disk errors and application program abends. The information can be retrieved and reported; other sources are the system log (SYSLOG) and the DB2 job output listing. However, in some cases, only the program can provide enough detail to identify the exact nature of problem.

The standardized procedure can be incorporated into application programs or 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 information recorded includes the date, time, authorization ID, terminal ID or job name, application, view or table affected, error code, and error description. Reports by time and by authorization ID can be produced and reviewed daily.

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, a history log should be kept of any DB2 utility that updates data, particularly REPAIR. Reports of updates by utilities (obtained through SMF customized reporting or a user-developed program) should be produced and reviewed regularly.

How Can DB2 Recover Data After Failures?

DB2 provides extensive methods of recovering data after a subsystem, media, or program failure. If there is a subsystem failure, a restart of DB2 automatically restores the integrity of the data by backing out uncommitted changes and completing the processing of committed changes. If there is a media failure (such as physical damage to a data storage device), the RECOVER utility can recover data to the current point. If there is a program error, the RECOVER utility can recover data to a specific log record or to a specific image copy. For detailed

information and recommendations, see “Recovering Table Spaces and Data Sets” on page 4-141.

The recovery methods depend on adequate image copies of table spaces to be recovered and on 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. It should be run daily and reviewed to assure 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.

If IMS, CICS, or a remote DBMS 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, there is an automatic record to provide an audit trail. The new release number is recorded by the 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” that 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 under “Chapter 5-5. Improving Resource Utilization” on page 5-73. The three tools mentioned below can be particularly useful:

1. 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 5-76.
2. The accounting trace is similar to the audit trace 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 X-177.

3. The performance trace is also pertinent. It can provide an enormous amount of detail, and is usually used for investigating particular problems. It also is described in “DB2 Trace” on page X-177.

Chapter 3-7. A Sample Security Plan for Employee Data

This chapter shows one way in which authorization IDs, implicit privileges, granted privileges and authorities, and the audit trace can all be used together to enforce a security plan. For example, suppose that our sample enterprise, the Spiffy Computer Company, decides upon 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 paid to other employees. They most particularly should not be able to update those values, for themselves or others.
- Managers have legitimate reasons for knowing the compensations paid to people who report to them. And there must be someone who can make changes to salary data.

From that compromise, 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 surely be different.

The Security Objectives: The following is the list of security objectives for Spiffy's security plan:

- 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 an auxiliary 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. (It turns out that this objective cannot be fully achieved. At the very least, some ID must sometime exercise powers reserved to SYSADM authority, and at that time it 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, it is possible to create a view of employee data for every employee reporting to a manager—even if he manages more than one department. Such a view requires altering department table DSN8410.DEPT by adding a column to contain managers' IDs:

```
ALTER TABLE DSN8510.DEPT
   ADD MGRID CHAR(8) FOR SBCS DATA NOT NULL WITH DEFAULT;
```

Every manager should have the SELECT privilege on a view created as follows:

```
CREATE VIEW DEPTMGR AS
   SELECT * FROM DSN8510.EMP, DSN8510.DEPT
      WHERE WORKDEPT = DEPTNO
      AND MGRID = USER;
```

To What ID Is the SELECT Privilege Granted?

We assume that virtually every employee of the Spiffy Computer Company uses a terminal, has a TSO logon ID and a personal password, and can access DB2I or QMF. The security planners can take two approaches to granting privileges:

1. Grant privileges to personal IDs and revoke them if the user of the ID leaves the company or transfers to another position. We call this the *individual* approach.
2. Create RACF groups and grant privileges to the group IDs, with the intention of never revoking them. When a personal ID needs those privileges, connect it to the group; disconnect it when its user leaves or transfers. We call this the *functional* approach. Another example of grouping takes place when many authorization IDs are translated into a single outbound ID. This is an implied approach rather than explicit.

The functional approach is probably more convenient in the following situations:

- There are many different privileges 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.
- There are several users that need the same set of privileges. Again, the set probably constitutes a business function.
- The privileges are given with the grant option, or they allow users to create objects that must persist after their original owners leave or transfer. In both cases, it can be undesirable to have to revoke the privileges. The revokes cascade to other users, and to change ownership it can be necessary to drop objects and re-create them.

What, then, of 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.

But in this case, there is only one privilege 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 have 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. Or, both approaches can be used simultaneously; some departments could be represented by their managers' personal IDs, others 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 personal 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 questions remaining 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 answered those questions as follows (and, again, we remind you that this 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 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 being 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 below, under:

"Actions at the Central Server Location" on page 3-138 and
"Actions at Remote Locations" on page 3-138.

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. One such entry might look like this:

Table 46. 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 systems 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 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 system must take the actions described below. (For a system other than DB2 for OS/390, the actions might be somewhat different; check the documentation for the product you are using. But in any case, the remote system must satisfy the requirements 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 attach requests to that location. Such an entry might look like this:

Table 47. 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 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. Such an entry might look like this:

Table 48. 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 being sent to the LU name specified in the LINKNAME column. All other IDs are translated to CLERK before being sent to that LU.

Table 49. 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 there is an audit record of 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. A summary of the reports is made every two months, and scanned for any 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; and all rows *except* those for members of the payroll department. Members of Payroll Operations have the 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, there is an auxiliary table, 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.

Commission amounts at Spiffy Computer Company are a separate problem. They 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, of course, is embodied in an application plan, the “commission program,” which must be 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 on several other tables as well.

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, they 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 includes these other controls as well:

- The employee ID in the auxiliary 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 ID.
- The employee ID in the auxiliary table is also a primary key for that table, so its values are unique. Because of that, in any one operating period (say, 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.
- There is 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 reject automatically any insert or update 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 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 Section 2 of *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:

- There are several privileges needed—the privileges on the views and probably also the EXECUTE privilege on the application plan for the commission program.
- There are several members of the department who 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 (possibly their only secondary ID). 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 3-101. DB2USER could retain ownership of the group, or more probably assign the ownership to an ID 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 used by managers, and by the 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 DSN8D51A, 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 were pleased that they did not have to grant that authority to any ID at all. For regular operations, it was enough to have 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.

A number of planners pointed out that database DSN8D51A contained several other tables (actually, all of those described in “Appendix A. DB2 Sample Tables” on page X-7). Some of the planners suggested putting the payroll tables into another database. That way, those with access to DSN8D51A could not access them. Others suggested that it might be convenient to have an administrative ID that could access those fully. That observation suggested the functional approach to privileges. Though 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 finally called for a RACF group that had:

1. DBCTRL authority over DSN8D51A
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 not only in the employee and payroll update tables, but 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, which is responsible for DB2 security. That does not mean that only IDs connected to DB2OWNER exercise all that authority, grant privileges on every plan and package, 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 granted to them specifically; but they can run utilities and examine the output data sets, or grant privileges that would allow other IDs to access data. Thus it is somewhat inconvenient for them to access sensitive data, but it is 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 cannot run the plans they bind without being specifically granted the EXECUTE privilege.

The Employee Table Owner

We said earlier that Spiffy Computer Company can never fully achieve its stated objective that only a manager can ever retrieve an employee's data record. In planning the views and GRANT statements described above, 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 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.
- There are several other payroll programs that 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 accesses 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.

Section 4. Operation and Recovery

Chapter 4-1. Basic Operation

The information under this heading, up to “Running IMS Application Programs” on page 4-18, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

The simplest elements of operation for DB2 for OS/390 are described in this chapter; they include:

Normal operation also requires more complex tasks. They are described in:

- “Chapter 4-2. Monitoring and Controlling DB2 and Its Connections” on page 4-23, 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 4-3. Managing the Log and the Bootstrap Data Set” on page 4-83, which describes the roles of the log and the bootstrap data set in preparing for restart and recovery.
- “Chapter 4-4. Restarting DB2 After Termination” on page 4-99, which tells what happens when DB2 terminates normally or abnormally and how to restart it while maintaining data integrity.
- “Chapter 4-5. Maintaining Consistency Across Multiple Systems” on page 4-109, which explains the two-phase commit process and the resolution of indoubt units of recovery.
- “Chapter 4-6. Backing Up and Recovering Databases” on page 4-123, which explains how to prepare for recovery as well as how to recover.

Recovery after various types of failure is described in:

- “Chapter 4-7. Recovery Scenarios” on page 4-155
- “Chapter 4-8. Recovery from BSDS or Log Failure During Restart” on page 4-221

Operating a Data Sharing Group: Although many of the commands and operational procedures described here are the same in a data sharing environment, there are some special considerations, which are described in Chapter 6 of *Data Sharing: Planning and Administration*. In particular, there are the following things to consider when 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).

Use of these commands is described in “Chapter 4-2. Monitoring and Controlling DB2 and Its Connections” on page 4-23. For a full description of the commands available, see Chapter 2 of *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 (off-loads) 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 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 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
	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
	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 PROCEDURE	Starts a stored procedure that is stopped, or refreshes one that is cached
	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 the distributed data facility
	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
- An IMS terminal or program
- A CICS terminal
- A TSO terminal
- An APF-authorized program
- An IFI application program.
- The DB2 Commands window of the DB2 PM Workstation Online Monitor

From an MVS console or MVS application: 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 Section 2 of *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 (*sysid*)

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 4-16.

From an APF-authorized Program: As with IMS, DB2 commands can be passed from an APF-authorized program to multiple DB2 subsystems by the MGCRE (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. The CART token is required 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, you 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, call attachment facility (CAF) attaches, and 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 X-126.

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 *MVS/ESA 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 53 on page 3-18. 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 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 the following:

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 3-93. *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 via 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:

```
$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
DSNR005I - RESTART...COUNTS AFTER FORWARD RECOVERY
           IN COMMIT=nnnn, INDOUBT=nnnn
DSNR006I - RESTART...COUNTS AFTER BACKWARD RECOVERY
           INFLIGHT=nnnn, IN ABORT=nnnn
DSNR002I - RESTART COMPLETED
DSN9022I - DSNYASCP '-START DB2' 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 4-62. 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 4-107.

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.

To accomplish the check, compare the expanded JCL in the SYSOUT output against the correct JCL provided in *MVS/ESA JCL User's Guide* or *MVS/ESA 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.

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 4-4. Restarting DB2 After Termination” on page 4-99.

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 4-99.

The following messages appear:

```
DSNY002I - SUBSYSTEM STOPPING
DSN9022I - DSNYASCP '-STOP DB2' NORMAL COMPLETION
DSN3104I - DSN3EC00 - TERMINATION COMPLETE
```

Before DB2 can be restarted, the following message must also appear at 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 which 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 4-17
“Running IMS Application Programs” on page 4-18
“Running CICS Application Programs” on page 4-18
“Running Batch Application Programs” on page 4-19
“Running Application Programs Using CAF” on page 4-20.
“Running Application Programs Using RRSAF” on page 4-20

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 Section 2 of *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 do the following:

- 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(DSN8BH51) LIB ('prefix.RUNLIB.LOAD')
END
```

A TSO application program 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 4-8.

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 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 3-4. Controlling Access to a DB2 Subsystem” on page 3-63 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 Section 5 of *Application Programming and SQL Guide* for more information about application programs and DL/I batch. See *IMS/ESA 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.

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 3-4. Controlling Access to a DB2 Subsystem” on page 3-63 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 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)
subcommand
.
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 4-6. Backing Up and Recovering Databases” on page 4-123. For a explanation of backing out changes to data when a batch program run in the TSO background abends, see Section 5 of *Application Programming and SQL Guide*.
- (*ssid*) is the subsystem name or group attachment name.

General-use Programming Interface

Running Application Programs Using CAF

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 Section 6 of *Application Programming and SQL Guide*.

End of General-use Programming Interface

General-use Programming Interface

Running Application Programs Using RRSAF

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 do the following things:

- 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 4-58 for a description of how applications connect to DB2 using RRSAF. For an explanation of RRSAF's capabilities and how to use it, see Section 6 of *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 *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 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 50 summarizes the operational control that is available at the operator console or terminal.

Table 50. Operational Control Summary

Type of Operation	MVS Console	TSO Terminal	IMS Master Terminal	Authorized CICSTerminal
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 CICScommands	Yes[4]	No	No	Yes
Receive CICSattachment facility unsolicited output	No[3]	No	No	Yes[5]

Notes:

1. Except START DB2. To enter commands from IMS, prefix them with /SSR; from CICS, prefix with 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 4-21.
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 4-2. Monitoring and Controlling DB2 and Its Connections

The information under this heading, up to “Controlling IMS Connections” on page 4-49, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

“Chapter 4-1. Basic Operation” on page 4-7 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 DB2 Utilities” on page 4-33
“Controlling the IRLM” on page 4-34.

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 4-37
“Controlling TSO Connections” on page 4-38
“Controlling CICS Connections” on page 4-41
“Controlling IMS Connections” on page 4-49
“Controlling OS/390 RRS Connections” on page 4-58
“Controlling Connections to Remote Systems” on page 4-62.

“Controlling Traces” on page 4-79, tells you how to start and stop traces, and points to other books for help in analyzing their results.

A final section, “Controlling the Resource Limit Facility (Governor)” on page 4-81, tells how to start and stop the governor, and how to display its current status.

Examples of commands in this chapter do not necessarily illustrate all the available options. For the complete syntax of any command or utility, see *Command Reference* or *Utility Guide and Reference*.

Controlling DB2 Databases and Buffer Pools

DB2 databases are controlled by the following commands:

START DATABASE	Makes a database, or individual partitions, available. For its use, see “Starting Databases” on page 4-24.
DISPLAY DATABASE	Displays status, user, and locking information for a database. For its use, see “Monitoring Databases” on page 4-25.
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 4-31.

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 DSN8S51E in the database DSN8D51A:

```
-START DATABASE (DSN8D51A) SPACENAM (DSN8S51E) 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 53 on page 3-18). 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.

Starting an Object with a Specific Status

A database, table space, or index space can be started with a specific status that limits access to it. 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 an partitioned index or logical partitions of a type 2 nonpartitioned index. The started or stopped state of other partitions is unchanged.

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 make any of them unavailable by using the command STOP DATABASE. DB2 also can 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, this command starts all table spaces and index spaces in database DSN8D51A for read-only access.

```
-START DATABASE (DSN8D51A) SPACENAM(*) ACCESS(RO)
```

The system responds with this message:

```
DSN9022I - DSNTDDIS '-START DATABASE' NORMAL COMPLETION
```

Starting a Table Space or Index Space against 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 against 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, no matter what restrictions are in force against them, by using the ACCESS(FORCE) option of START DATABASE, as in:

```
-START DATABASE (DSN8D51A) SPACENAM (DSN8S51E) ACCESS(FORCE)
```

The command releases all restrictions for the named objects. These objects must be explicitly named in a list following the SPACENAM option.

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
******* 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. The following options are described more in this publication:
  - USE (4-27)
  - LOCKS (4-27)
  - LPL (4-28)

  See Chapter 2 of *Command Reference* for detailed descriptions of these and other options.
- *Dbname* is an 8-byte character string indicating the database name.
- STATUS is a combination of one or more of the following strings delimited by a comma. The maximum length of the string is 18 characters. Anything beyond 18 characters is truncated.

CHKP The object (a table space or a partition within a table space) is in the check pending state.

COPY The object (a table space or a partition within a table space) is in the copy pending state. An image copy is required for this object.

GRECP The object (a table space, table space partition, index space, index partition, or logical index partition) is in the group buffer pool recovery pending state. You'll see this only when DB2 is part of a data sharing group.

LPL The object (a table space, table space partition, index space, index partition, or logical index partition) has pages on a logical page list.

PSRCP The index space is in a page set recovery pending state.

RECP The object (a table space, table space partition, index space, index partition, or logical index partition) is in the recovery pending state.

If an asterisk (*) appears (RECP*), it indicates that a logical index partition is in RECP, but that the entire index is inaccessible to SQL requests.

RO The object (database, table space, table space partition, index space, index partition, or logical index partition) is started for read-only activity.

RW The object (database, table space, table space partition, index space, index partition, or logical index partition) is started for read and write activity.

STOP The object (database, table space, table space partition, index space, index partition, or logical index partition) is stopped.

STOPE The table space or index space was implicitly stopped because DB2 detected an invalid log RBA or LRSN in one of its pages. Message DSNT500I was issued when the error was detected and indicates which page is in error.

STOPP A stop is pending for the object (database, table space, table space partition, index space, index partition, or logical index partition).

UT The object (database, table space, table space partition, index space, index partition, or logical index partition) is started for utility processing only.

UTRO A utility is in process on an object (table space, table space partition, index space, index partition, or logical index partition) that allows only RO access.

UTRW A utility is in process on an object (table space, table space partition, index space, index partition, or logical index partition) that allows only RW access.

UTUT A utility is in process on an object (table space, table space partition, index space, index partition, or logical index partition) that allows only UT access.

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 name 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 like the following, which names partitions 2, 5, and 67 in table space SALES_HX in database DSN8D51A:

```
-DISPLAY DATABASE (DSN8D51A) SPACENAM (SALES_HX) PART(2,5,67) USE
```

DB2 returns a list similar to this one:

```
11:44:32 DSNT360I - ************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *        GLOBAL USE
11:44:32 DSNT360I - ************************************************
11:44:32 DSNT362I -     DATABASE = DBPARTS  STATUS = RW
                        DBD LENGTH = yyyy
11:44:32 DSNT397I -
NAME     TYPE PART STATUS           CONNID   CORRID       USERID
-------- ---- ---- ---------------- -------- ------------ ----------
SALES_HX TS   002  RW               BATCH    TSOUSER1     BAT1
 DB2NET.LUND0.143992156557=1 ACCESSING DATA FOR USIBMSTODB22
SALES_HX TS   005  RW               IMSA     0012IMSPSB01 BAT2
SALEX_HX TS   067  RO               BATCH    TSOUSER1     BAT1
******* DISPLAY OF DATABASE DSN8D51A ENDED       **********************
11:45:15 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 Section 3 of *Messages and Codes*.

Obtaining Information about Pages in Error

There are two ways that pages can be in error: logically and physically.

- 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 DASD 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 *error page range*. 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.

A program that tries to read data from a page listed on the LPL or error page range 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. For example:

```
-DISPLAY DB(DBFW8401) SPACENAM(*) LPL
```

Output similar to the following is produced:

```
DSNT360I = ************************************************************
DSNT361I = *  DISPLAY DATABASE SUMMARY
           *    GLOBAL LPL
DSNT360I = ************************************************************
DSNT362I =     DATABASE = DBFW8401  STATUS = RW
               DBD LENGTH = 8066
DSNT397I =
NAME     TYPE PART STATUS             LPL PAGES
-------- ---- ---- ------------------ ------------------
TPFW8401 TS    01  RW,LPL             000000-000004
TPFW8401 TS    02  RW
TPFW8401 TS    03  RW
TPFW8401 TS    04  RW
TPFW8401 TS    05  RW
TPFW8401 TS    06  RW
TPFW8401 TS    07  RW
⋮
TPFW8401 TS    16  RW
ICFW8401 IX    01  RW,LPL             000000,000003
ICFW8401 IX    02  RW
ICFW8401 IX    03  RW
ICFW8401 IX    04  RW
⋮
ICFW8401 IX    14  RW
ICFW8401 IX    15  RW
ICFW8401 IX    16  RW
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 *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 up:

- 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.

- Run the RECOVER utility on the object.

  The only exception to this is when a logical partition of a type 2 nonpartitioned index has both LPL and RECP status. If you want to recover the logical partition using RECOVER INDEX with the PART keyword, you must first use the command START DATABASE to clear the LPL pages.

- Run 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
RECOVER INDEX
RECOVER TABLESPACE, *except*:
    RECOVER TABLESPACE...PAGE
    RECOVER TABLESPACE...ERROR RANGE
REPAIR with the SET statement
REPORT

LPL Combined with Other Restrictive Statuses: If the DISPLAY DATABASE command shows a partitioned table space or index to have both LPL and RECP statuses, run the RECOVER utility to remove them.

If the DISPLAY DATABASE command shows that a logical partition of a type 2 nonpartitioned index has both LPL and RECP statuses, you have two options:

- Run the RECOVER INDEX utility on the entire nonpartitioned index
- Issue START DATABASE for the index and then run the RECOVER INDEX utility using the PART option on the logical index parts to remove the RECP status.

If the DISPLAY DATABASE command shows the nonpartitioned index as having both LPL and PSRCP statuses, run the RECOVER INDEX utility to recover the entire nonpartitioned index and remove both the PSRCP and LPL statuses.

Resetting Recover Pending Status: If you must make an object available to DB2 without removing pages from the LPL, you can do it in either of these ways:

- Execute the REPAIR utility with the NORCVRPEND option on the object.
- Start the object with access (FORCE).

Resetting recover pending status without removing pages from the LPL is not recommended.

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)
```

might display a list such as this:

```
11:44:32 DSNT360I - ************************************************
11:44:32 DSNT361I - *  DISPLAY DATABASE SUMMARY
11:44:32            *          GLOBAL
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 Section 3 of *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 an unpartitioned type 2 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. STOP DATABASE with the PART option does not close data sets associated with logically partitioned spaces, however. To close these data sets, you must execute STOP DATABASE without the PART option.

The AT(COMMIT) option of STOP DATABASE determines when objects are stopped. The AT(COMMIT) option is required to interrupt threads that are bound with RELEASE(DEALLOCATE), especially in situations with high thread reuse.

If you specify AT(COMMIT), DB2 takes over access to an object when all jobs release their claims on it, and 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 in order to stop user-defined databases or the work file database. The only exception to this is when the pending lock is held by an indoubt unit of recovery. In this case, the STOP DATABASE command terminates and the stop status of each object that was stopped by the command is backed out. A DSNI003I message tells you the command was unable to stop an object. You must resolve the indoubt unit of recovery and run the job again.

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 (*database name*)
: Stops a database, and closes all of the data sets of the table spaces and index spaces in the database.

-STOP DATABASE (*database name1*, *database name2*)
: 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 (*database name*) 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 (*database name*) SPACENAM (*table space or index 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 (*database name*) SPACENAM (*table space or index 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.

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 *Command Reference* for descriptions of the options you can use with this command. See "Tuning Database Buffer Pools" on page 5-49 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:

```
DSNB401I   BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 10
DSNB402I   VIRTUAL BUFFERPOOL SIZE = 1000 BUFFERS
             ALLOCATED       =     1000   TO BE DELETED   =        0
             IN-USE/UPDATED  =      200
DSNB403I   HIPERPOOL SIZE = 100000 BUFFERS, CASTOUT = YES
             ALLOCATED       =   100000   TO BE DELETED   =        0
             BACKED BY ES    =        0
DSNB404I   THRESHOLDS -
             VP SEQUENTIAL         = 80   HP SEQUENTIAL         = 80
             DEFERRED WRITE        = 50   VERTICAL DEFERRED WRT = 10
             PARALLEL SEQUENTIAL   = 50   ASSISTING PARALLEL SEQT= 80
DSNB405I   HIPERSPACE NAMES - @001SSOP
             IOP SEQUENTIAL        =  0
```

See Chapter 2 of *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 DB2 Utilities

You can run DB2 utilities against databases, table spaces, index spaces, and partitions.

DB2 utilities are classified into 2 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 Section 2 of *Utility Guide and Reference*, and the stand-alone utilities are described in Section 3 of of that same publication.

Starting Online Utilities: To start a DB2 utility, prepare an appropriate set of JCL statements for a utility job and include DB2 utility statements in the input stream for that job.

Controlling Online Utilities: The following commands for monitoring and changing DB2 utility jobs are described in Chapter 2 of *Command Reference*.

ALTER UTILITY	Alters parameter values of a running REORG utility
DISPLAY UTILITY	Displays the status of utility jobs
TERM UTILITY	Terminates a utility job before its normal completion

Read-only utilities are allowed on an object started RO. All utilities are allowed on an object started RW; however, only DB2 utilities are permitted to access an object started UT. If an object is started with no other indication, its status is RW. To change the status of an object, start it with the new status using the ACCESS option. For example:

```
-START DATABASE (DSN8D51A) ACCESS(RO)
```

Section 2 of *Utility Guide and Reference* shows, for each online utility, what classes it drains and the other utilities or SQL operations with which it is compatible.

Stand-Alone Utilities: The following stand-alone utilities can be run only by means of MVS JCL:

DSN1CHKR
DSN1COPY
DSN1COMP
DSN1PRNT
DSN1SDMP
DSN1LOGP
DSNJU003 (change log inventory)
DSNJU004 (print log map)

Most of the stand-alone utilities can be used while DB2 is running, but for consistency of output it is recommended that DB2 be stopped first because these utilities do not have access to the DB2 buffer pools.

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 up 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 *Data Sharing: Planning and Administration* for more information about configuring IRLM in a data sharing environment.

The following MVS commands can be used to monitor and control the IRLM:

	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 dynamically the maximum amount of CSA storage or the number of trace buffers used for this IRLM
#	MODIFY *irlmproc*,SET,CSA=nnn	Sets dynamically the maximum amount of CSA storage that this IRLM can use for lock control structures
#	MODIFY *irlmproc*,SET,TRACE=nnn	Sets dynamically the maximum number of trace buffers used for this IRLM
	MODIFY *irlmproc*,STATUS	Displays the status for the subsystems on this IRLM
#	MODIFY *irlmproc*,STATUS,*irlmx*	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,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
	START *irlmproc*	Starts the IRLM
	STOP *irlmproc*	Stops the IRLM normally
#	TRACE CT,OFF,COMP=*irlmx*	Stops IRLM tracing
#	TRACE CT,ON,COMP=*irlmx*	Starts IRLM tracing for all subtypes (DBM,SLM,XIT,XCF)
#	TRACE CT,ON,COMP=*irlmx*,SUB=(*subname*)	Starts IRLM tracing for a single subtype

Starting the IRLM

The IRLM must be available when DB2 starts, or 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 irlmx 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 expereriences a failure that involves the IRLM.

When you start the IRLM manually, you can generate a dump to collect diagnostic information.

Monitoring the IRLM Connection

To display the status of all subsystems connected to an IRLM, use this MVS command:

```
MODIFY irlmproc,STATUS
```

In MVS, MODIFY is abbreviated by F; you can enter F *irlmproc*,STATUS.

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, the following message appears:

```
DXR105E irlmx 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
```

The system responds with this message:

```
DXR124E irlmx ABENDED VIA MODIFY COMMAND
```

DB2 abends. An IMS subsystem using the IRLM does not abend, and can be reconnected.

IRLM does exploit 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 linklist. 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 will de-register from ARM during normal shutdowns using:

- a STOP command

• a MODIFY irlmproc,ABEND,NODUMP command
• auto-stop of IRLM.

Use the MODIFY command listed above to FORCE the DMBSs using the IRLM
down and stop IRLM without having it automatically restarted by ARM. IRLM will
de-register DB2 from ARM before DB2 abends to prevent ARM from restarting DB2
and IRLM if using the auto-start feature.

Monitoring Threads

The DB2 command DISPLAY THREAD displays current information about the status of threads. It includes information about threads processing locally or those
| processing distributed requests. DISPLAY THREAD can also be used to monitor
| 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 via a network with another system and performing work on behalf of that system.
- An inactive database access thread is one that is connected via 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 “The Command ARCHIVE LOG” on page 4-88.

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 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 *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
```

More information about how to interpret this output can be found in the sections describing the individual connections and in the description of message DSNV404I in Section 3 of *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 4-37 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
- 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 4-17.

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 up (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 51 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 51. Differences in DISPLAY THREAD Information for TSO and Batch

Connection	Name	AUTHID	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 list displayed, see the description of message DSNV404I in Section 3 of *Messages and Codes*.

The following command displays information about TSO and CAF threads, including those processing requests to or from remote locations:

```
-DIS THD(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
```

Figure 67. DISPLAY THREAD Showing TSO and CAF Connections

Description of output:

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 attach facility application.

5 This is a TSO batch application originating thread.

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.

Detailed information for assisting the console operator in identifying threads involved in distributed processing may be found in “Monitoring Threads” on page 4-37.

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 *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 Section 2 of *Installation Guide*. For details on the CICS transaction entry parameters, see *CICS/MVS Resource Definition (Online)* . For details on the DB2 SYSOPR authority, see “Chapter 3-2. Controlling Access to DB2 Objects” on page 3-13.

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. The command you use to start the attachment facility depends on which level of CICS you are running.

CICS Version 3.3 and earlier:

```
DSNC STRT x
```

CICS Version 4.1 and later:

```
DSNC STRT xx,ssid
```

x or *xx* names a particular resource control table suffix (DSNCRCT*x* or DSN2CT*xx*). With the attachment shipped with Version 4.1, you can also specify a DB2 subsystem ID 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 Section 2 of *Installation Guide* .

Messages

For information about messages that appear during connection, see Section 3 of *Messages and Codes*. These messages begin with "DSNC," unless you are running with the attachment shipped with CICS Version 4.1. 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/MVS Resource Definition (Macro)*.

In releases after CICS 4.1, the CICS attachment facility enables the INDOUBTWAIT function to resolve indoubt units of recovery automatically. See *CICS/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/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 resolves the unit of work when CICS is reconnected to DB2.

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:

```
DSNV406I -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 Section 3 of *Messages and Codes*.

Recovering Indoubt URs Manually

Under some circumstances, CICS cannot resolve indoubt units of recovery. When this happens, message DSNC001I, DSNC034I, DSNC035I, or DSNC036I is sent to the user-named CICS destination, specified in the resource control table (RCT). (These messages begin with “DSN2” if you are running with the CICS attachment that is shipped with CICS Version 4.1.)

To recover an indoubt unit, issue the following command:

```
-RECOVER INDOUBT (connection-name) ACTION (COMMIT|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 4-113.

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 4-166. For information on the two-phase commit process, as well as indoubt units of recovery, see “Consistency with Other Systems” on page 4-109.

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 is described in “Recommendations for RCT Definitions” on page 5-132. For more information on specifying the THRDS parameter, see Section 2 of *Installation Guide*.

At any time during execution, thread subtasks can be created. If the following message is displayed:

```
DSNC017I 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 Section 2 of *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.

```
DSNC DISPLAY STATISTICS destination
```

This is an example of the output for the DSNC DISPLAY (STATISTICS) command:

```
DSNC014I   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
DSNC020I   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.

ONE-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.

TWO-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 52 summarizes these differences.

Table 52. 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:

```
-DIS THD(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
```

Figure 68. DISPLAY THREAD Showing CICS Connections

Description of output:

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.

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. There are two ways to disconnect an application that we describe 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. To terminate, that thread must be processing 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 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/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/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 *Command Reference* or, in the IMS library, to *IMS/ESA Operator's 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 DB2 connection was active when IMS is shut down
- In response to the command /START SUBSYS *sysid*, where *sysid* 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
  ```

  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 53 summarizes these differences.

Table 53. 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
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:

```
-DIS THD(applid)
```

```
  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
```

Figure 69. DISPLAY THREAD Showing IMS Connections

Description of output:

1 This is a message-driven BMP.

2 This thread has completed sign-on processing, but a DB2 plan has not been allocated.

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/ESA Operator's 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. To terminate, that thread must be processing in DB2.

General-use Programming Interface

Displaying Indoubt Units of Recovery

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 4-4. Restarting DB2 After Termination” on page 4-99. This chapter describes only the operational steps used to list and recover indoubt units in relatively simple cases.

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 INDOUBT THREADS - 920
COORDINATOR               STATUS      RESET URID         AUTHID
SYS3                      INDOUBT           00017854FF6B ADMF001
 V449-HAS NID= SYS3.400000000 AND ID= 0001BMP255
BATCH                     INDOUBT           00017854A8A0 ADMF001
 V449-HAS NID= DSN:0001.0 AND ID= RUNP10
BATCH                     INDOUBT           00017854AA2E ADMF001
 V449-HAS NID= DSN:0002.0 AND ID= RUNP90
BATCH                     INDOUBT           0001785CD711 ADMF001
 V449-HAS NID= DSN:0004.0 AND ID= RUNP12
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 Section 3 of *Messages and Codes*.

End of General-use Programming Interface

General-use Programming Interface

Recovering Indoubt Units

To recover an indoubt unit, issue the following command:

```
-RECOVER INDOUBT (imsid) ACTION (COMMIT|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 4-113.

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

General-use Programming Interface

Duplicate Correlation IDs

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 which represents the number of IMS schedulings 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 which 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 4-161.
- 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 sysid
  ```

  To purge the RRE, give one of these commands:

  ```
  /CHANGE SUBSYS sysid RESET
  /CHANGE SUBSYS sysid 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. batch message processing (BMP) program

   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 Section 2 of *Installation Guide* for further details.

General-use Programming Interface

Monitoring the Activity on Connections

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 list displayed, see the description of message DSNV404I in Section 3 of *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 4-65.

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 sysid
```

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)
```

Figure 70 on page 4-57 shows four examples of the output that might be generated when an IMS /DISPLAY SUBSYS command is issued.

- Example 1 shows that the DSN subsystem is not yet connected. The message DSNM003I was issued by the IMS attachment facility.
- Example 2 shows a connected status. The message DSNM001I was issued by the IMS attachment facility.
- Example 3 shows both a /STOP SUBSYS command and a /DISPLAY SUBSYS command. The output that was displayed in response to /DISPLAY SUBSYS shows a stopped status. The message DSNM002I was issued by the IMS attachment facility.

Example 1

```
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
```

Example 2

```
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
```

Example 3

```
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
```

Example 4

```
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 70. Sample Output from IMS /DISPLAY SUBSYS Command

- Example 4 shows a connected status, and the region ID (1) is included. Use the REGID (*pst#*) and PROGRAM (*pstname*) values to correlate the output of the IMS /DIS SUBSYS command to the LTERM involved.

End of General-use Programming Interface

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 sysid
```

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 *sysid* 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 sysid - RC=z
```

That message uses the following reason codes (RC):

Code	Meaning
A	IMS/TM is terminating normally (for instance, /CHE FREEZE|DUMPQ|PURGE). 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 *sysid* 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 signon exit routine.
AUTH SIGNON	Provides a user ID, an ACEE, and optionally, one or more secondary authorization IDs to be associated with the connection. Invokes the signon 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 Section 6 of *Application Programming and SQL Guide* .

General-use Programming Interface

Connecting to OS/390 RRS Using RRSAF

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 Section 3 of *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 4-113.

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 4-109.

End of General-use Programming Interface

General-use Programming Interface

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
```

Figure 71. DISPLAY THREAD Showing RRSAF Connections

Description of output:

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.

Disconnecting Applications from DB2

There is no way to 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. The thread must be processing in DB2 to terminate.

End of General-use Programming Interface

Controlling Connections to Remote Systems

The information under this heading, up to "Using NetView to Monitor Errors in the Network" on page 4-76, is General-use Programming Interface and Associated Guidance Information, as defined in "Notices" on page xi.

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"
"Monitoring Connections to Other Systems" on page 4-63, which describes the use of the following commands:

- DISPLAY LOCATION
- DISPLAY THREAD
- CANCEL THREAD
- VARY NET,TERM (VTAM command)

"Monitoring and Controlling Stored Procedures" on page 4-72
"Using NetView to Monitor Errors in the Network" on page 4-76
"Stopping DDF" on page 4-78

Related Information: The following topics in this book contain information about distributed connections:

"Chapter 5-11. Monitoring and Tuning in a Distributed Environment" on page 5-315
"Resolving Indoubt Units of Recovery" on page 4-113
"Failure of a Database Access Thread" on page 4-194

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 4-5. Maintaining Consistency Across Multiple Systems" on page 4-109 for information about resolving indoubt threads.

Using the START DDF command requires authority of SYSOPR or higher. The command causes the following messages to appear:

```
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.

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 Section 2 of *Installation Guide*.

Monitoring Connections to Other Systems

Two DB2 commands give you information about distributed threads. Use DISPLAY LOCATION to get summary statistics about distributed threads. Use DISPLAY THREAD to get more detailed information about specific threads.

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 protocal 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   RD   *        0 028.DBAA  SYSOPR           0035       10
    V445-USIBMSY.SYEC81A.AD4F183D5303=10 ACCESSING DATA FOR
         <SYEC81A>:SYEC81A
       BATCH    TR   *        5 BKH2C     SYSADM   BKH2    000D       1 3
    V444-DB2NET.LUND0.9F6D9F459E92=1 3  ACCESSING DATA AT
    V446-USIBMSTODB22:LUND1 4
    V447-LOCATION            SESSID       A ST TIME
    V448-USIBMSTODB22        4019:446 5   V R3 9015611253116
   DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DIS THD' NORMAL COMPLETION
```

Example Label Description

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 *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 *1* (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 application 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 Section 3 of *Messages and Codes* .

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 *Command Reference*for more information on the DETAIL keyword.

For example issue:

```
-DIS THD(*) 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.

```
-DIS THD(*) LOC(*) DET
  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
```

Example Label Description

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. The value *W* indicates that the thread is suspended in DB2 and is waiting for notification from VTAM that the event has completed. A value *V* would indicate 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, Section 3 of *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 72. 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.

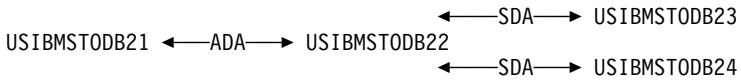


Figure 72. Example of a DB2 transaction involving four sites. ADA refers to DRDA access, SDA to DB2 private protocol access

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” on page 4-70.

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
```

Example Label Description

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 4-71.

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 4-113 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/luwid HAS BEEN CANCELLED
```

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 *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 Section 4 of *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. Take the following 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 73 on page 4-72, 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 73. 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 74, 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 74. 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 Stored Procedures Environment" on page 4-74
- "Obtaining Diagnostic Information About Stored Procedures" on page 4-76

For more information about stored procedures, see Section 6 of *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 that have been accessed by DB2 applications:

```
-DISPLAY PROCEDURE

DSNX940I csect - DISPLAY PROCEDURE REPORT FOLLOWS-
PROCEDURE  MODULE    STATUS    ACTIVE  MAXACT  QUEUED  MAXQUE  TIMEOUT
USERPRC1   MODULE1   STARTED        0       1       0       1        0
USERPRC2   MODULE2   STOPQUE        0       2       5       5        3
USERPRC3   MODULE3   STARTED        2       2       0       6        0
USERPRC4   MODULE4   STOPREJ        0       1       0       1        0
DISPLAY PROCEDURE REPORT COMPLETE
```

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:

```
DSNV401I - DISPLAY THREAD REPORT FOLLOWS -
DSNV402I - ACTIVE THREADS - 176
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
BATCH    SP       3 RUNAPPL      SYSADM   PL01AP01 001D    43
 V429 CALLING PROCEDURE=PROC1, LOAD MODULE=LMPROC1
      PROC=V51ASPAS, ASID=0029, WLM_ENV=
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I - DSNVDT '-DISPLAY THREAD' 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.

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=V51AWLM1 SUBSYSTEM TYPE: DB2
```

The output tells you that WLMENV1 is available, so WLM can schedule stored procedures for execution in that environment.

Refreshing the Stored Procedures Environment

Depending on what has changed in a stored procedures environment, you might need to perform one or more of these tasks:

- Refresh the definition of a stored procedure.

  Do this when you change the definition of a stored procedure in catalog table SYSPROCEDURES.

- 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.

To refresh the definition of a stored procedure in catalog table SYSPROCEDURES, issue the commands -STOP PROCEDURE and -START PROCEDURE. To validate the contents of one or more SYSPROCEDURES rows as you refresh them, specify:

```
-START PROCEDURE(procedure-name)
or
-START PROCEDURE(partial-name*)
```

When you use either of these forms of -START PROCEDURE, DB2 checks values in the corresponding rows of SYSPROCEDURES. If any rows contain invalid values, DB2 issues an error message, then looks for cached rows for *procedure-name* or *partial-name**. If DB2 finds any cached rows, it uses the cached information to process stored procedure requests. See Chapter 2 of *Command Reference* for the complete syntax of -START PROCEDURE and -STOP PROCEDURE.

The method that you use to perform these other 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 Stored Procedures Environment” on page 4-74 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 75 on page 4-77 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 75. Alerts-Static Panel in NetView. DDF errors are denoted by the resource name AS (application server) and AR (application 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 brings up the Recommended Action for Selected Event panel shown in Figure 76.

```
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 [A]          AS [B]
            +--------+   +--------+
 DOMAIN     |  RQST  |---|  SRVR  |
            +--------+   +--------+

USER    CAUSED - NONE

INSTALL CAUSED - NONE

FAILURE CAUSED - SNA COMMUNICATIONS ERROR:
                     RCPRI=0008 RCSEC=0001              [A]
                FAILURE OCCURRED ON RELATIONAL DATA BASE USIBMSTODB21
       ACTIONS - I008 - PERFORM PROBLEM DETERMINATION PROCEDURE FOR REASON
                 CODE [C] 00D31029                  [B]
                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 76. Recommended Action for Selected Event Panel in NetView. In this example, the AR (USIBMSTODB21) is reporting the problem, which is affecting the AS (USIBMSTODB22).

Figure Label Description

A 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.

B 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.

C DB2 reason code. For information about DB2 reason codes, see Section 4 of *Messages and Codes*. For diagnostic information, see Section 4 of *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 4-113 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.

General-use Programming Interface

Controlling the DB2 Trace

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 *Command Reference* . For more information about the trace output produced, see “Appendix D. Interpreting DB2 Trace Output” on page X-107. 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 *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 Section 2 of *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/ESA 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/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 which 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 *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 5-76 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 Section 2 of *Installation Guide*.

Chapter 4-3. 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 DASD data set called the *active log*. When the active log is full, DB2 copies its contents to a DASD or tape data set called the *archive log*. That process is called *off-loading*. This chapter describes:

For information about the physical and logical records that make up the log, see "Appendix C. Reading Log Records" on page X-81. 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 4-109.)

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 77.

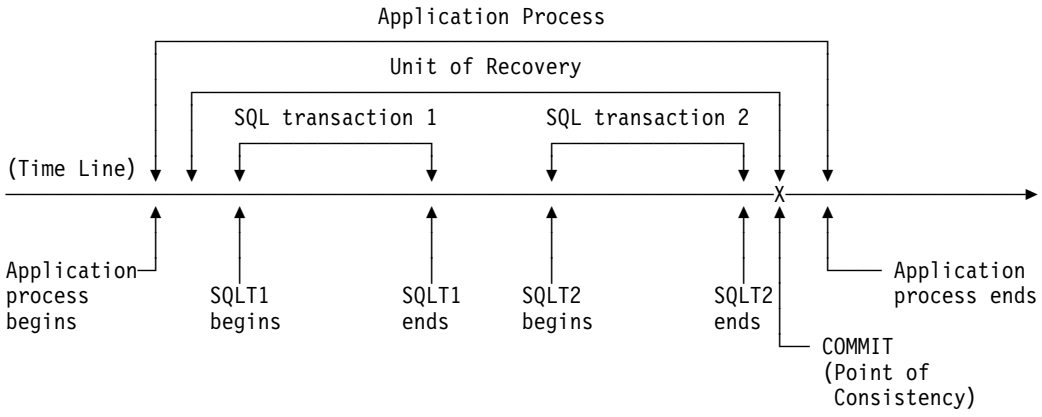


Figure 77. A unit of recovery within an application process

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 and DB2 can not commit. Not until the amount is added to account B are they consistent again. 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 78. The SQL ROLLBACK statement, and deadlocks and timeouts (reported as SQLCODE -911, SQLSTATE 40001) cause the same events.

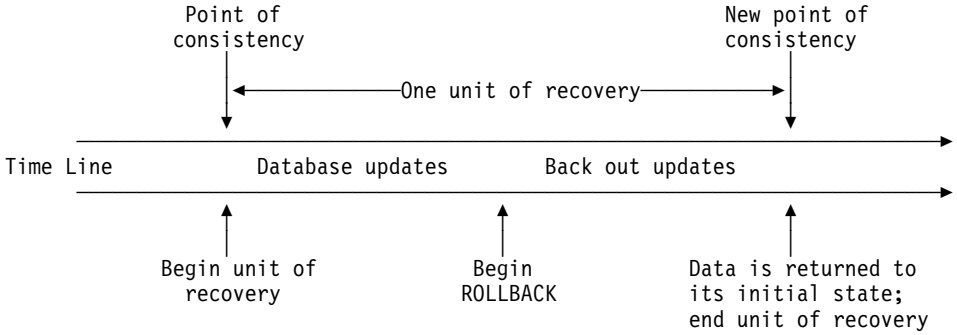


Figure 78. Unit of Recovery (ROLLBACK)

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 Section 2 of *Installation Guide* .

Creation of Log Records

The process of logging is shown schematically in Figure 79 on page 4-85. 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 *Data Sharing: Planning and Administration* for more information.)
4. The CIs are written to a set of predefined DASD *active log data sets*, which are used sequentially and recycled.
5. As each active log data set becomes full, its contents are automatically *off-loaded* to a new *archive log data set*.

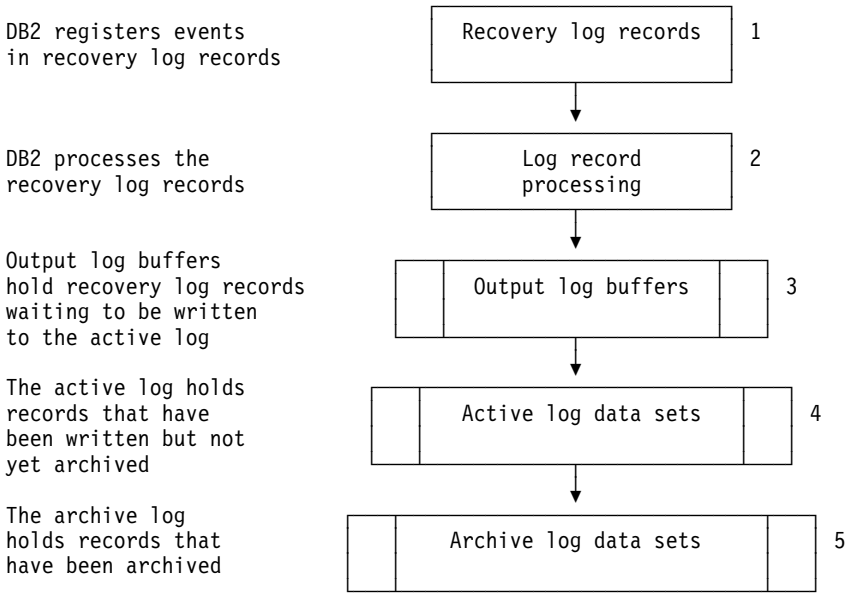


Figure 79. The Logging Process

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 (Off-Loading)

The process of copying active logs to archive logs is called *off-loading*. The relation of off-loading to other logging events is shown schematically in Figure 80.

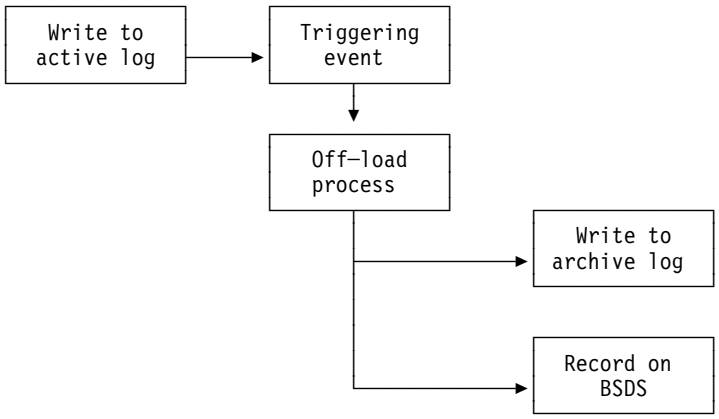


Figure 80. Off-Load Process

Triggering Off-Load

An off-load 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.

Off-load 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. Off-load 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. If all active logs become full, DB2 stops processing until off-loading occurs and issues this message:

  ```
  DSNJ111E - OUT OF SPACE IN ACTIVE LOG DATA SETS
  ```

The Off-Load Process

During the process, DB2 determines which data set to off-load. Using the last log RBA off-loaded, 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 off-load and halts processing until the off-load is completed. If the off-load 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 4-171.

When an active log is ready to be off-loaded, a request can be sent to the MVS console operator to mount a tape or prepare a DASD 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 off-load 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 off-load. In that case, if the allocation is for the first copy of dual archive data sets, the off-load 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 during Off-load: The following messages are sent to the MVS console by DB2 and the off-load 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 off-load 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 off-loading, 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 off-load task has ended:

  ```
  DSNJ139I LOG OFFLOAD TASK ENDED
  ```

Interruptions and Errors while Off-loading: Here is how DB2 handles the following interruptions in the off-loading process:

- The command STOP DB2 does not take effect until off-loading is finished.
- A DB2 failure during off-load causes off-load to begin again from the previous start RBA when DB2 is restarted.
- Off-load handling of read I/O errors on the active log is described under “Active Log Failure” on page 4-171, or write I/O errors on the archive log, under “Archive Log Failure” on page 4-175.
- An unknown problem that causes the off-load 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 4-90.

The Command ARCHIVE 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 off-load 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 off-load, and updates the BSDS with a record of the off-load. 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 Off-Loading: 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 off-loaded. Unlike the QUIESCE utility, ARCHIVE LOG MODE(QUIESCE) does not force all changed buffers to be written to DASD 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 off-load 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 *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 Section 2 of *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, as shown below.

```
-ARCHIVE LOG MODE(QUIESCE) TIME(60)
```

The above command allows for a quiesce period of up to 60 seconds before archive log processing occurs.

Attention

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, since 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 off-load 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 off-load 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 off-load process, such as a problem with allocation or tape mounting. If the active logs cannot be off-loaded, 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 off-load:

```
-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 off-loading. If the off-load fails again, you must fix the problem that is causing the failure before the command can work.

End of General-use Programming Interface

Archive Log Data Sets

Archive log data sets can be placed on standard label tapes or DASD 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 DASD 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 4KB. (For more information, see information about installation panel DSNTIPA in Section 2 of *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 DASD 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 Section 2 of *Installation Guide* for more information.

It is not possible to specify specific volumes for new archive logs. If allocation errors occur, off-loading is postponed until the next time off-loading 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 multi-volume data sets.

Archiving to DASD 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 DASD and the secondary copy go to tape.

This feature increases recovery speed without using as much DASD, and the tape
can be used as a backup.

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 off-load 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 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 that a copy of the BSDS is placed 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 DASD Volumes: All archive log data sets allocated on DASD must be cataloged. If you choose to archive to DASD, 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 DASD, you receive message DSNJ072E each time an archive log data set is allocated, although the DB2 subsystem still catalogs the data set.

If the unit name reflects DASD, the archive log data sets cannot extend to another volume.

If you use DASD, make the primary space allocation (both quantity and block size) large enough to contain all of the data coming from the active log data sets. That minimizes the possibility of unwanted MVS B37 or E37 ABENDs during the off-load process. Primary space allocation is set with the PRIMARY QUANTITY field of the DSNTIPA installation panel.

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 DASD 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 DASD 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 DASD. 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.

Insure 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.

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 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, it 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. Give 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 4-94.

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 4-95.

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 DASD, the BSDS copy is a separate file which could reside on a separate volume.

For better off-load 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 off-load 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 Section 3 of *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 Section 3 of *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, since 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 4-13). 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 4-2. Monitoring and Controlling DB2 and Its Connections” on page 4-23.

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 4-6. Backing Up and Recovering Databases” on page 4-123 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 DASD 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 Section 2 of *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 4-94. 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 4-92 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 on page 4-96. 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 on page 4-96.
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 4-113.

3. Reissue 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 on page 4-96.

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	SYSUTILX
	SCT02	SYSLGRNX
	SPT01	
DSNDB06	SYSCOPY	SYSPLAN
	SYSDBASE	SYSSTATS
	SYSDBAUT	SYSSTR
	SYSGPAUT	SYSUSER
	SYSGROUP	SYSVIEWS
	SYSPKAGE	

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 Section 3 of *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 Section 3 of *Utility Guide and Reference* .

     You should also consider increasing the value of MAXARCH. For information, see information about installation panel DSNTIPA in Section 2 of *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 DASD, 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 Section 3 of *Utility Guide and Reference*.

Chapter 4-4. 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 4-5. Maintaining Consistency Across Multiple Systems" on page 4-109 and "Chapter 4-6. Backing Up and Recovering Databases" on page 4-123. This chapter includes the following topics:

"Chapter 4-5. Maintaining Consistency Across Multiple Systems" on page 4-109 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 6 of *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 54.

Table 54. 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 are ended.
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:

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 4-111.

At the end of the fourth phase recovery is complete, 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. If your DB2 subsystem has the UR checkpoint option enabled, DB2 generates console message DSNR035I and trace records for IFCID 0313 to inform you about long running URs. The UR checkpoint option is enabled at installation time, through field UR CHECK FREQ on panel DSNTIPN. See Section 2 of *Installation Guide* for more information about enabling the UR checkpoint option.

For an example of the messages that are written to the DB2 console during restart processing, see "Messages at Start" on page 4-14.

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 4-102.
   - 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 DASD 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 4-177.
   - 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 Section 2 of *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 DSNJ001I, 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 DASD. It is possible for a unit of recovery to be declared complete before all database modifications are written to DASD.

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 Section 2 of *Installation Guide*.

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 (and phase 4 backward log recovery in the case of multiple systems) 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. 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 4-183.
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 indoubt units of recovery.

   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 DASD; 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 DASD, 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 DASD 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 DASD 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 4-102), 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 completes processing by reversing all changes performed for inflight or in-abort units of recovery. 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.
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 DASD, 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 reverse the changes and facilitate media recovery. They are written under all circumstances, even when:
   - The DASD version of the data did not need to be reversed.
   - The page set has pages are on the LPL.
   - An I/O error occurred on the DASD version of the data.
   - The DASD version of the data could not be allocated or opened.
4. Writes pages to DASD as the need for buffers demands it.
5. Finally, writes to DASD all modified buffers that have not yet been written.

6. Issues message DSNR006I, which summarizes the number of remaining in-abort or inflight units of recovery. The number for each should be zero, because all processing for them should have completed.
7. Marks the completion of each 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 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 after a failure.

When DB2 stops abnormally, MVS determines whether MVS failed, too, and where DB2 should be restarted. It then restarts DB2 appropriately.

You must have DB2 installed with a command prefix scope of S to take advantage of automatic restart. See Section 2 of *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 name of the DB2 subsystem. For a non data-sharing DB2, the ELEMENT 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 instructions on defining automatic restart policies, see *MVS/ESA 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 type 2 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.

There are different ways to correct down level page set errors. See “Recovery from Down-Level Page Sets” on page 4-183 for more information.

How to Defer Restart Processing

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.
- 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.

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.

Restarting with Conditions

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.

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 Section 3 of *Utility Guide and Reference*. For information on data sharing considerations, see Restarting a DB2 member with Conditions in Chapter 6 of *Data Sharing: Planning and Administration*.

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.

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 4-143.

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 Section 3 of *Utility Guide and Reference* for information about that utility.

Chapter 4-5. 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 DBMS 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.

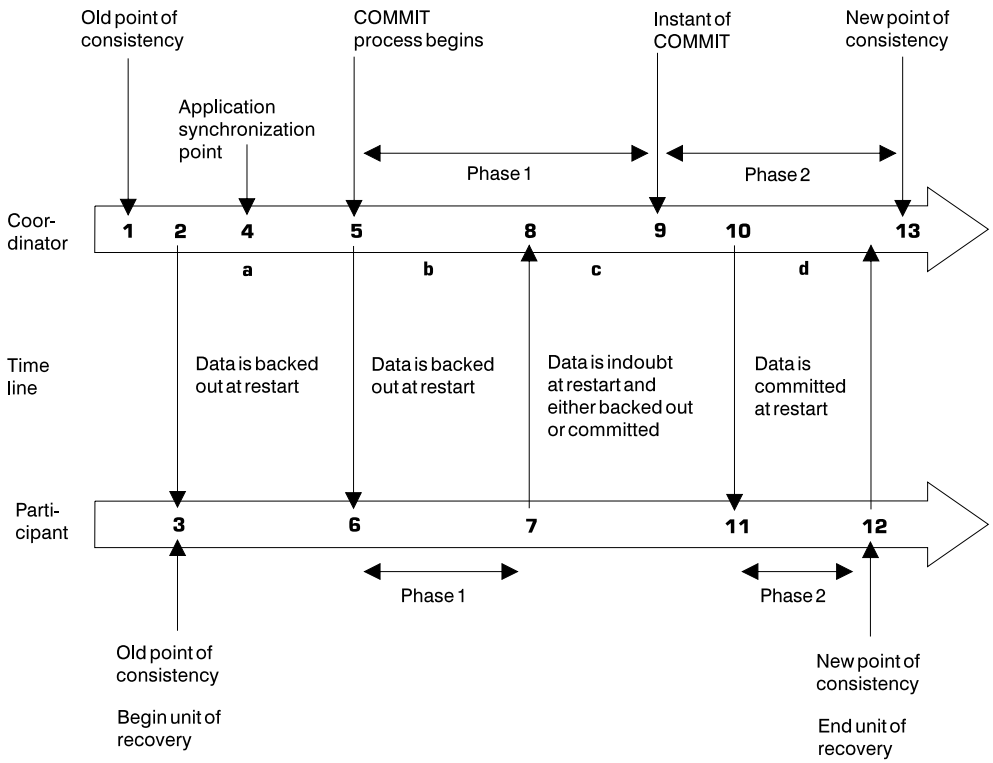


Figure 81. Time Line Illustrating Commit Coordinated with Another Subsystem

Illustration of Two-Phase Commit

Figure 81 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 81 on page 4-110 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.

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, 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 4-101.

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 4-111).

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 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 4-113.)

Phase 4: Backward Log Recovery

As described in “Phase 4: Backward Log Recovery” on page 4-104 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.

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 4-248.

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 the newly added message - DSNR036I for unresolved UR encountered during a checkpoint. This message might occur to remind operators of existing indoubt threads. See section 2 of Installation Guide for details.

Attention

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. Since 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 indoubts. 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 4-49.

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 4-166.

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.

Attention

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 4-37 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 abort 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 4-211.

Resetting an Indoubt Thread's Status

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 an 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 data base changes and to release data base 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 may 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 to 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 82 DB2A is the participant for an IMS transaction, but becomes the coordinator for the two application servers (AS1 and AS2), DB2B, and its respective DB2 servers (DB2C, DB2D AND DB2E).

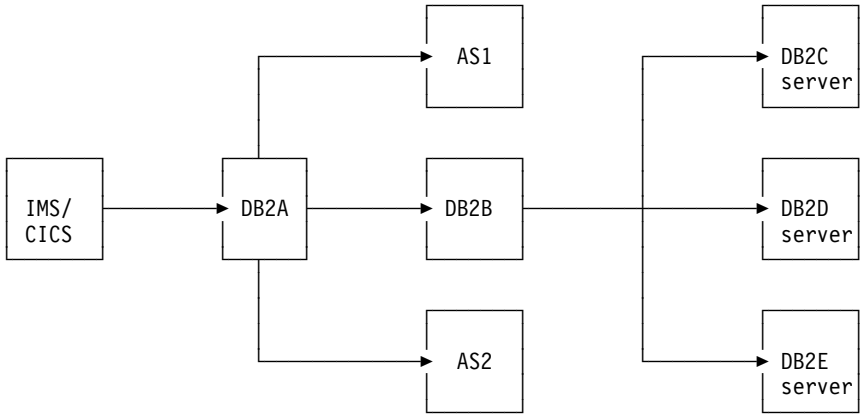


Figure 82. Illustration of multi-site unit of work

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

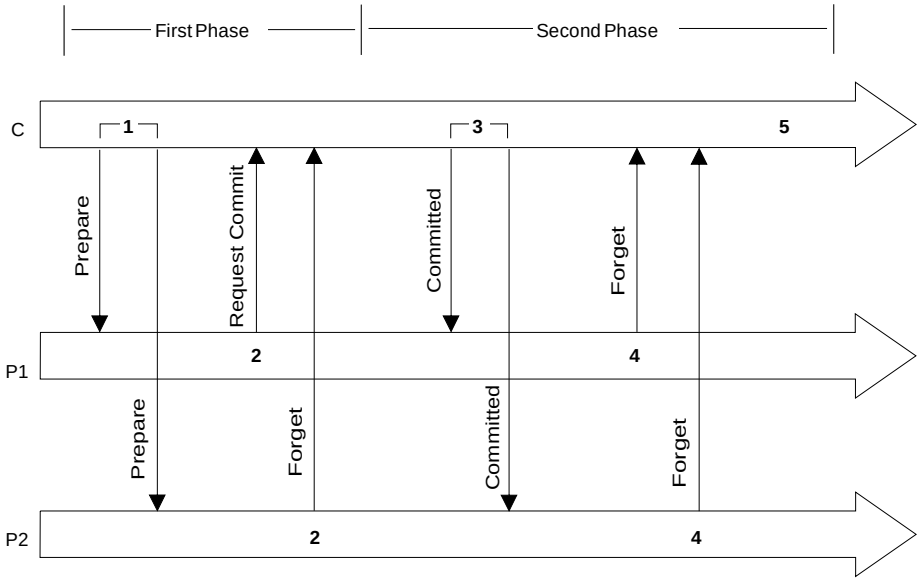


Figure 83. Illustration of multi-site update. C is the coordinator; P1 and P2 are the participants.

Figure 83 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 (P1) 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 (P2) 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 acknowledgement 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 4-6. 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, 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 *Command Reference* and *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 off-site recovery in case of a disaster.

To improve recovery capability in the event of a DASD 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 DASD volumes. These concepts are described in "Establishing the Logging Environment" on page 4-84.

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 4-124
- "Preparing for Recovery" on page 4-124
- "Making Backup and Recovery Plans that Maximize Availability" on page 4-128
- "What Happens during Recovery" on page 4-125
- "How to Find Recovery Information" on page 4-130
- "Preparing to Recover to a Prior Point of Consistency" on page 4-131
- "Preparing to Recover the Entire DB2 Subsystem to a Prior Point" on page 4-133

Considerations for Recovering Distributed Data

Using distributed data, no matter where a unit of work originates, it 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 their members, 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 so closely coordinated on separate subsystems. After that, recovery planning is a matter of agreement among database administrators at separate locations.

Since 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.

Preparing for Recovery

To ensure that a table space can be recovered to a particular point, there must be a copy of it at some earlier state. That is called a backup copy. The DB2 recovery log contains a record of all changes made to the table space. If DB2 fails, it can recover the table space by restoring the backup copy and applying the log changes to it from the point of the backup copy.

The DB2 catalog and directory table spaces must be copied at least as frequently as the most critical user table spaces. 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 recreate 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 immediately. That gives you a starting point from which to recover. Use the COPY utility, as described in Section 2 of *Utility Guide and Reference*.

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, made Monday.

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, Thursday, and Friday Mornings: You make another incremental image 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 Section 2 of *Utility Guide and Reference*. The utility restores the table space from the full image copy made Monday and the incremental image copies made Tuesday through 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.

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 84 on page 4-126 shows an overview of the recovery process. To recover a table space, the RECOVER utility uses these items:

- A full image copy; that is, a complete copy of the table space
- 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 created since the most recent image copy.

The RECOVER utility uses information in the SYSIBM.SYSCOPY catalog table to:

- Restore the table space with the data in the most recent full image copy (appearing, in Figure 84 on page 4-126, at X'40000').
- Apply all the changes summarized in any later incremental image copies. (There are two in Figure 84, X'80020' and X'C0040'.)
- Apply all changes to the table space that are registered in the log, beginning at the log RBA of the most recent image copy. (In Figure 84, X'C0040', that address is also stored in SYSIBM.SYSCOPY.)

If the log has been damaged or discarded, you can recover to a particular point in time by limiting the range of log records to be applied by the RECOVER utility.

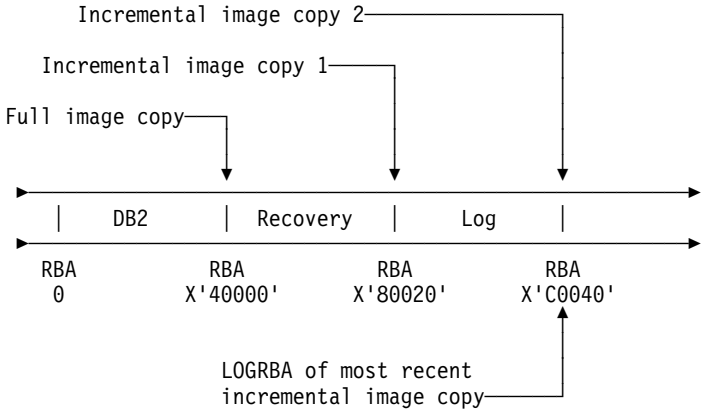


Figure 84. Overview of DB2 Recovery. The figure shows one complete cycle of image copies; SYSIBM.SYSCOPY can record many complete cycles.

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 during execution of the utility, so DB2 does not place the table space in copy pending status.

If you use LOG(YES), and log all updates, then an image copy is not required for data integrity. However, taking an image copy makes the recovery process more efficient. The process is still 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.

Use the CHANGELIMIT option of the COPY utility to let DB2 determine when an image copy should be performed 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.

In determining how many complete copy and log cycles to keep, you are guarding against damage to a volume containing an important image copy. Typically, you will want a retention period of at least two full cycles. When you take a scheduled full image copy of a table space, the latest full image copy and at least one other previous copy exists, plus the log created since the earliest image copy. For further security, keep records for three or more copy cycles.

A Recovery Cycle Example

Table 55 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 55. 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 recover to the latest point of consistency, 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 RECOVER INDEX in Section 2 of *Utility Guide and Reference* for more information.

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 DASD 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 DASD. 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 table spaces 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 table space 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 table space 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 4-94 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 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 four gigabytes of data, and you can read about two gigabytes of data per hour, then you should make copies after every four gigabytes of data written. You can use the COPY option SHRLEVEL CHANGE or DFSMS Concurrent Copy to make copies while transactions and batch jobs are running. You should also make copy after jobs that make large numbers of changes.

Determine the right characteristics for your logs:

- If you have enough DASD 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.

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 Section 5 of *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 *MVS/ESA 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 are some major factors that influence the speed of DB2 startup:

- DB2 checkpoint interval

  This is the factor that has the most influence on the speed of DB2 startup. This is the number of log records that DB2 writes between successive checkpoints. This value is controlled by the DB2 subsystem parameter LOGLOAD. The default of 50000 results in the fastest DB2 startup time in most cases.

- 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.

- 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 Section 5 of *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.

  SYSCOPY also contains entries with the same kinds of log RBAs recorded by the utilities QUIESCE, REORG, LOAD, RECOVER TOCOPY, and RECOVER TORBA (or TOLOGPOINT). For a summary of the information contained in the DB2 catalog tables, see Appendix D of *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 DSNDB01.SYSCOPY are quiesced or copied, a SYSCOPY record is created for each table space. 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 table space 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 4-154 describes the process.

Reporting Recovery Information

You can use the REPORT utility in planning for recovery. REPORT provides information necessary for recovering a table space. 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 Section 2 of *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 suitable 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 QUIESCE utility to determine whether the data is in an exception status. You cannot quiesce a table space that is in check pending, copy pending, or recovery pending status. If the return code from QUIESCE is 8, investigate the problems with the table space before copying it. See “Violations of Referential Constraints” on page 4-192 and information on the COPY and RECOVER utilities in Section 2 of *Utility Guide and Reference* for instructions on resetting those statuses.

Also, if QUIESCE encounters an I/O error, if the table space has a write error range, or if the table space has a pending deferred restart, QUIESCE issues a warning message. In these cases, the table space is quiesced, but cannot be copied.

Step 2: Copying the Data

You can copy the data and also establish a point of consistency, 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.

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 DASD dump.

  You can use the CONCURRENT option of the COPY utility to make a backup with DFSMS Concurrent Copy that is recorded in the DB2 catalog. For guidance in using this option see *Utility Guide and Reference*.

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 table spaces. Typically, you name all the table spaces in a table space set, which you want to recover to the same moment to avoid referential integrity violations. The following statement quiesces two table spaces in database DSN8D51A:

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

QUIESCE writes changed pages from the table spaces to DASD. The catalog table SYSIBM.SYSCOPY records the current RBA and the timestamp of the quiesce point. At that point, neither table space contains any uncommitted data. A row with ICTYPE Q is inserted into SYSCOPY for each table space quiesced. (Table spaces

DSNDB06.SYSCOPY, DSNDB01.DBD01, and DSNDB01.SYSUTILX, are an exception: their information is written in the log.)

QUIESCE allows concurrency with many other utilities; however, it does not allow concurrent updates until it has quiesced all specified table spaces. Depending upon the amount of activity, that can take considerable time. Try to run QUIESCE when system activity is low.

Preparing to Recover the Entire DB2 Subsystem to a Prior Point

If there are problems during a restart of DB2, you might want to reset the entire system to a point of consistency. You can prepare a point of consistency 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. Copy SYSLGRNX and SYSCOPY last. Install job DSNTIJIC creates image copies of the DB2 catalog and directory table spaces. See Section 2 of *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 4-243 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.

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 table spaces
- 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 4-197 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 X-104 and “Log Capture Routines” on page X-68.

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 85 is an overview of the process of preparing to bring DB2 up at a recovery site.

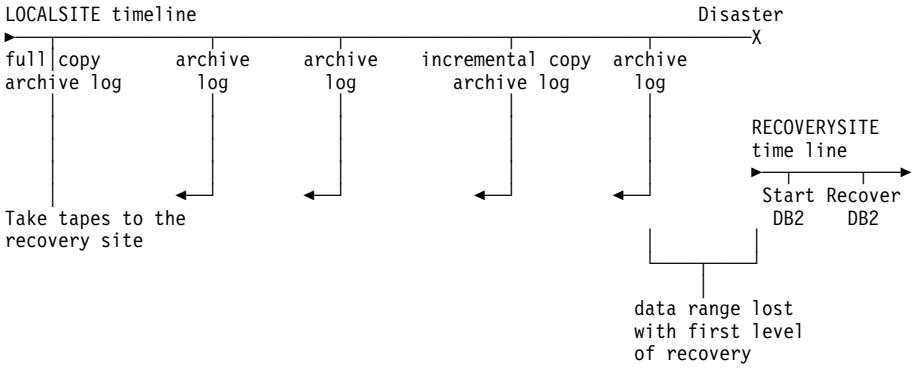


Figure 85. Preparing for Disaster Recovery. The information you need to recover is contained in the copies of data (including the DB2 catalog and directory) and the archive log data sets.

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. 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 which you have installed using the RECOVERYSITE option.[8] For information about making multiple image copies, see COPY in Section 2 of *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.

- 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 "The Command ARCHIVE LOG" on page 4-88.

     # Do not use dual logging for disaster recovery from a remote site. If the
     # first copy of an archive becomes unreadable, then DB2 requests a
     # second copy and waits until the second copy is mounted. The
     # requesting function fails if the second copy is not available.

     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.
  5. Record this activity at the recovery site when the EXPORT backup and list are received.

---

[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.

- 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. Document your backups.
  3. Send backups and corresponding documentation to the recovery site.
  4. 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 4-197.

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 4, make a full image copy of the DB2 directory and catalog** using installation job DSNTIJIC. See Section 2 of *Installation Guide* for DSNTIJIC information.

  If you did not do this during installation or migration, use the COPY utility, described in Section 2 of *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 Section 2 of *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 4-84 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, either:

- rename the data sets that contain the table spaces 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 table space containing inconsistent data, the RECOVER utility cannot resolve the data inconsistency problem. However, RECOVER can be used to resolve the inconsistency if you have an older image copy of that table space taken before the problem occurred. Use the MODIFY utility with the RECOVERY option to delete image copy information from SYSIBM.SYSCOPY. Image copy data sets must be deleted separately. To retain a particular image copy for use by RECOVER, you can use the MODIFY utility with the ICDATE of that image copy. This also retains any later image copies. See Section 2 of *Utility Guide and Reference* for information about the MODIFY utility.

- **Maintain consistent referential structures.**

  A referential structure is a set of tables 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. REPORT TABLESPACESET 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.

  See “Chapter 2-3. Maintaining Data Integrity” on page 2-19 for information on designing referential structures to help maintain the consistency of your data.

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 4-94.

- **Do not make an image copy of a table space containing inconsistent data.**

  If you use the COPY utility to make an image copy of a table space containing inconsistent data, the RECOVER cannot recover a problem involving that table space unless you have an older image copy of that table space taken before the problem occurred. You can run DSN1COPY with the CHECK option to determine whether intra-page data inconsistency problems exist on table

spaces before making image copies of them. If you are taking a copy of a catalog or directory table space, 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 *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, creating a consistency problem. 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 the COPY or RECOVER 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 Section 2 of *Utility Guide and Reference*.

Running RECOVER Jobs in Parallel

You can schedule jobs with the RECOVER utility in two ways:

- Schedule one RECOVER job to process the entire table space. This option provides none of the benefits of parallel operation.
- 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 DASD 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 DASD are read entirely in parallel.
  – A data set controlled by DFSMShsm is first recalled. It then resides on DASD 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.

  In general, scheduling RECOVER jobs in parallel significantly reduces the job time.

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 X-81.

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 Table Spaces and Data Sets

You can use the COPY utility to copy data from a table space to an MVS sequential data set on DASD 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 offsite recovery operations. It does not copy indexes, as that operation is unnecessary; indexes can be recovered from recovered table spaces.

Use the MERGECOPY utility to merge several image copies.

| 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 that 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.

| 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 separately from the LOAD or REORG utility because it decreases the time that your table spaces are unavailable.

Related Information: For guidance in using COPY and MERGECOPY and making image copies during LOAD and REORG, see Section 2 of *Utility Guide and Reference*.

Backup 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.

Recovering Table Spaces and Data Sets

You can recover objects in either of these ways:

- If you made backup copies using the DB2 COPY 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.
- If you made backup copies using a method outside of DB2's control, such as with DSN1COPY or the Concurrent Copy function of DFSMS, 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:

- Restore the most current full image copy
- Apply changes recorded in later incremental image copies
- Apply later changes from the archive or active log.

RECOVER can act on:

- An entire table space, or list of table spaces
- A specific data set within a table space (usually a partition)
- A single page
- A page range within a table space that DB2 has found in error
- An index
- 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 TABLESPACE and RECOVER INDEX, see Section 2 of *Utility Guide and Reference*.

Not every recovery operation requires RECOVER; see also

“Recovering Error Ranges for a Work File Table Space” on page 4-143
“Recovering the Work File Database” on page 4-142
“Recovering Data to a Prior Point of Consistency” on page 4-144.

A Caution about DASD Dump: Be very careful when using DASD dump and restore for recovering a data set. DASD dump and restore can make one data set inconsistent with DB2 subsystem tables in some other data set. Use DASD 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 4-131.

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 DASD 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 DASD error has really been corrected or not. 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 DASD 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 table space as part of a list of table spaces. See the description of RECOVER TABLESPACE in Section 2 of *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 table spaces 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 4-105.

Point-in-time recovery: Full recovery of the catalog and directory table spaces and indexes is strongly recommended. However, if you need to plan for point-in-time recovery of the catalog and directory, then one method of creating a point of consistency is to:

1. Quiesce all catalog and directory table spaces in a list, except for DSNDB06.SYSCOPY and DSNDB01.SYSUTILX.
2. Quiesce DSNDB06.SYSCOPY.

   We recommend that you quiesce DSNDB06.SYSCOPY in a separate utility statement; when you recover DSNDB06.SYSCOPY to its own quiesce point, it

contains the ICTYPE = 'Q' (quiesce) SYSIBM.SYSCOPY records for the other catalog and directory table spaces.

3. Quiesce DSNDB01.SYSUTILX in a separate job step.

If you need to recover to a point in time, recover DSNDB06.SYSCOPY and DSNDB01.SYSUTILX to their own quiesce points, and recover other catalog and directory table spaces to their common quiesce point. The catalog and directory objects must be recovered in a particular order. See Section 2 of *Utility Guide and Reference*for more information.

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 TABLESPACE utility to restore directory table spaces DSNDBD01 or SYSUTILX, or catalog table space SYSCOPY will fail with abend 00E40119. Instead of using the RECOVER TABLESPACE 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 TABLESPACE utility with the LOGONLY option to apply updates from the log records to the recovered table spaces.
3. Recover the indexes with the RECOVER INDEX utility.
4. Take a full image copy of the table spaces 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 4-146
- “Backing Up and Restoring Data with Non-DB2 Dump and Restore” on page 4-147
- “Using RECOVER to Restore Data to a Previous Point in Time” on page 4-147.

The following considerations apply to all methods for backing up and restoring data:

Be Aware of Table Space Sets: If you restore a table space to a prior state, restore all related tables to the same point to avoid inconsistencies. The table spaces that contain related tables are called 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 table spaces that belong to a single table space set and then restore those table spaces which are related. However, if there are related table spaces that belong to more than one table space set or there are table spaces that are logically related in application

programs of which DB2 is not aware, you are responsible for identifying all the table spaces on your own.

Recovering Indexes: Indexes do not have to be backed up the way data is. Recovery of indexes does not rely on information in the log. Use RECOVER INDEX to restore the indexes after the data has been 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.
- 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*.I0001.A00*n*.

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.
   - 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, recreate

the table definition using CREATE TABLE. To recreate 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 4-151 to recreate the entry in the DB2 catalog.

See Section 3 of *Utility Guide and Reference* for more information about DSN1COPY and DSN1PRNT.

Recover 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 DFSMS 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 Section 3 of *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 *vol-ser*.

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 table space creation to the chosen log point. This assumes you have not modified SYSLGRNX for that table space.

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, use SHRLEVEL(REFERENCE) to establish consistent points for TOCOPY recovery. Copies made with SHRLEVEL(CHANGE) do not copy data at a single instant, because changes can occur as the copy is made. A later RECOVER TOCOPY operation can produce inconsistent data.

| 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 table space or table space set, and then quiesce them using the QUIESCE utility. This allows RECOVER TORBA to recover the table spaces to the quiesce point with minimal use of the log.

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

Point-in-time recovery does not restore related indexes to a consistent point, as does recovery to the current point in time, when the indexes are already consistent. Therefore, after a recovery made with TOCOPY or TORBA, indexes are placed in *recovery pending* status. See information about RECOVER in Section 2 of *Utility Guide and Reference* for information about resetting this status.

Point-in-time recovery can cause table spaces to be placed in *check pending* status if they have table check constraints or referential constraints defined on them. When recovering tables involved in a referential constraint, you should recover all the table spaces involved in a constraint. This is the *table space set*. To avoid setting check pending, you must do both of the following:

- Recover the table space 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 4-192.

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. The details of data compression are described in “Compressing Data in a Table Space or Partition” on page 2-63. For important information on loading and compressing data see the description of LOAD in Section 2 of *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. See “WITH RESTRICT ON DROP Clause” on page 2-97 for more information about this clause.

Limitations of the Procedures

These procedures do not reclaim a dropped table in a segmented table space. Because of the way space is reused for segmented table spaces, there is no simple, predictable procedure to recover these tables. (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 4-151.

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

In order to perform this procedure, you will need a full image copy or a DSN1COPY file that contains the data from the dropped table.

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.

      - Field HPGOBID is four bytes long and contains the DBID in the first two bytes and the PSID in the last two bytes.
      - Field PGSOBD 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 recreate 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 work data set back to 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 Section 3 of *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, perhaps by executing SELECT statements to use the table.
9. Make a full image copy of the table space. See "Copying Table Spaces and Data Sets" on page 4-139 for more information about the COPY utility.
10. Recreate the objects that are dependent on the table.

    As explained in "Implications of Dropping a Table" on page 2-134, 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 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 may 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 4-150.
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 recreate 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 1 and 6 as the input records for the translation file (SYSXLAT). For more information about DSN1COPY, see Section 3 of *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 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 Table Spaces and Data Sets” on page 4-139 for more information about the COPY utility.

12. Recreate the objects that are dependent on the table.

    See step 10 of “Recovery of an Accidentally Dropped Table” on page 4-150 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 4-150.
2. Recreate 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. To allow DSN1COPY to access the DB2 data set, stop the table space with the following command:

   ```
   -STOP DATABASE(database name) SPACENAM(tablespace-name)
   ```

4. Find the “new” DBID, PSID, and OBIDs by querying the DB2 catalog as described in step 6 of “User-Managed Data Sets” on page 4-151.
5. Run DSN1COPY using OBIDXLAT and RESET options to perform the OBID translation and to copy the data from the full image copy data set or the DSN1COPY data set. Use the OBIDs you recorded from steps 1 and 4 as the input records for the translation file (SYSXLAT). For more information about DSN1COPY, see Section 3 of *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. Drop all “dummy” tables. The row structure does not match the definition, so these tables cannot be used.
8. Reorganize the table space to remove all rows from dropped tables.
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 Table Spaces and Data Sets” on page 4-139 for more information about the COPY utility.
12. Recreate the objects that are dependent on the table.

    See step 10 on page 4-151 of “Recovery of an Accidentally Dropped Table” on page 4-150 for more information.

Discarding SYSCOPY and SYSLGRNX Records

Use the MODIFY utility to delete obsolete records from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX.

1. Follow these steps of the procedure described under “Locating Archive Log Data Sets to Delete” on page 4-95:

   a. Resolve Indoubt Units of Recovery on page 4-95.
   b. Find the Startup Log RBA on page 4-96.
   c. Find the Minimum Log RBA Needed on page 4-96. 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.

Considerations for Table Space Set Recovery: To determine a valid quiesce point for the table space set, use the procedure for determining a RECOVER TORBA value. See RECOVER in Section 2 of *Utility Guide and Reference* for more information.

Chapter 4-7. 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 irlmx 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 4-35.)
- Start DB2. (For instructions, see “Starting DB2” on page 4-13.)
- 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 4-41.)

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 4-35.)
4. Start DB2. (See “Starting DB2” on page 4-13.)
5. Restart IMS or CICS.
    a. IMS automatically connects and resynchronizes when it is restarted. (See “Connecting to the IMS Control Region” on page 4-50.)
    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 4-41.)

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.

DASD Failure

Problem: A DASD hardware failure occurs, resulting in the loss of an entire unit.

Symptom: No I/O activity for the affected DASD 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 DASD 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 DASD 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 parmlib member or by issuing the MVS command

   ```
   SETIOS MIH,DEV=devnum,IOTIMING=mm:ss.
   ```

   For more information on the I/O timing facility, see *MVS/ESA Initialization and Tuning Reference* and *MVS/ESA 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 DASD unit must be disconnected for repair, all databases and table spaces on all volumes in the DASD unit must be stopped.

3. Select a spare DASD pack and use ICKDSF to initialize from scratch a DASD 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 DASD unit.

4. Issue this MVS console command. *yyy* is the new unit address.

   ```
   VARY yyy,ONLINE
   ```

5. To check DASD 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*.I0001.A00x CLUSTER NOSCRATCH

   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 following access method services commands for each one of them:

   DEFINE *catnam*.DSNDBC.*dbname.tsname*.I0001.A00x CLUSTER

   DEFINE *catnam*.DSNDBD.*dbname.tsname*.I0001.A00x DATA

   Detailed requirements for user-managed data sets are described in "Requirements for Your Own Data Sets" on page 2-69.

   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 TABLESPACE utility. Additional information and procedures for recovering data can be found in "Recovering Table Spaces and Data Sets" on page 4-141.

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 00 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 4-147.

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, specify 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 Section 3 of *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 91 on page 4-231.
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."
- "Resolution of Indoubt Units of Recovery" on page 4-161.
- "IMS Application Failure" on page 4-163.

Extended Recovery Facility (XRF) Toleration

DB2 can be used in an XRF recovery environment with IMS. All DB2-owned data sets (executable code, the DB2 catalog, user databases) must be on DASD 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. 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.

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 abended 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, since 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(COMMIT³ABORT) 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 4-53.)

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 4-53.)

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

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 COMMIT³ABORT
```

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 Section 3 of *Application Programming and SQL Guide*. For procedures to diagnose DB2 problems, rather than user errors, refer to Section 4 of *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:

Extended Recovery Facility (XRF) Toleration

DB2 can be used in an XRF recovery environment with CICS. All DB2-owned data sets (executable code, the DB2 catalog, user databases) must be on DASD 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. That can be done automatically, by using the facilities provided by CICS, or manually, by the system operator. 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 BSDS must be included as a protected resource, and the primary and alternate XRF processors must be included in the GRS ring.

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 Section 3 of *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 *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 Section 3 of *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 4-41. 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 Inability to 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 4-41.
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 has abended, 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, since 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): DSNC001I, DSNC034I, DSNC035I, or DSNC036I. (These messages begin with “DSN2” if they are from a CICS Version 4.1 release, or later).

System Action: The system action is summarized in Table 56:

Table 56 (Page 1 of 2). CICS Abnormal Indoubt Unit of Recovery Situations

Message ID	Meaning
DSNC001I	The named unit of recovery cannot be resolved by CICS because CICS was cold started. The CICS attachment facility continues the startup process
DSNC034I	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

Table 56 (Page 2 of 2). CICS Abnormal Indoubt Unit of Recovery Situations

Message ID	Meaning
DSNC035I	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
DSNC036I	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, since 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) 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

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/MVS Operations Guide* for details on how to use this program.

Step 3: Scan the DB2 log for entries related to a particular unit ofrecovery: 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 Section 3 of *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, since only DB2 resources are affected. However, CICS keeps details about the indoubt threads which 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 Section 3 of *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:

```
DSNC025I - 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 pseudoabends.
  - For option Q, the message is enqueued again and the transaction is abended.
  - For option A, the message is discarded and the transaction is abended.
- 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

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 4-8. Recovery from BSDS or Log Failure During Restart" on page 4-221.

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 off-loading 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 off-load 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 re-issues the message after each additional 5 percent of the data set space is filled. Each time the message is issued, the off-load process is started.

```
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 off-load is started. The DB2 subsystem then halts processing until an off-load 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 off-load is not waiting for a tape drive. If it is, mount a tape and DB2 will process the off-load 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 off-load task. This might solve the problem.

If this command doesn't solve the problem, you must determine the cause of the problem and then re-issue 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 off-load failures is corrected.

1. Use the DB2 -STOP 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 off-loads 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 Section 3 of *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 off-loaded 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 off-load.

For instructions on using the change log inventory utility, see Section 3 of *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 off-loaded or were marked stopped.

System Action: Continues in single mode until off-loading completes, then returns to dual mode. If the data set is marked “stopped,” however, then intervention is required.

System Programmer Action: Check that off-load 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 Section 3 of *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 off-load, off-load 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 off-load 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 off-load, 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 off-load 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 Section 3 of *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 off-loaded:

   a. Use 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 off-loaded (that is, if the value for High RBA Offloaded 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 off-loaded; 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 4-8. Recovery from BSDS or Log Failure During Restart” on page 4-221.

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 4-8. Recovery from BSDS or Log Failure During Restart” on page 4-221.

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 off-load 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 off-load 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 4-8. Recovery from BSDS or Log Failure During Restart” on page 4-221.

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, since 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 Off-load

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 off-load task has abended and you should consult Section 3 of *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 off-loading 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 off-load 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 off-load this RBA range is made the next time off-load is triggered.

– The active log does not wrap around; if there are no more active logs, data is not lost.

Operator Action: Ensure that off-load 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 doesn't 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 Section 2 of *Utility Guide and Reference* for more information about using the RECOVER utility.

Problem 4 - Insufficient DASD Space for Off-load Processing

Symptom: While off-loading the active log data sets to DASD, DB2 off-load 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 off-load. If the off-load cannot compete in single archive mode, the active log data sets cannot be off-loaded, and the status of the active log data sets remains NOTREUSEABLE. Another attempt to off-load the RBA range of the active log data sets is made the next time off-load 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 off-load 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 off-loaded 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 DASD 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 DASD 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 off-load.

- 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 DASD device. This condition is normally accompanied by MVS ABEND message IEC032I.

  To solve the problem, make space available on the DASD 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 Section 2 of *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.

BSDS Failure

For information about the BSDS, read “Managing the Bootstrap Data Set (BSDS)” on page 4-92. 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 4-179.

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 off-load (message DSNJ125I).

See Section 3 of *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
```

eeii since it printed incorrectly.þþ>

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 4-241.
- 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 4-244).

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 off-loading 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 DASD, the BSDS is allocated on any available DASD. 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 off-loading it), then locate an earlier copy of the BSDS from an earlier off-load.

3. Rename any *damaged* BSDS by using the access method services ALTER command with the NEWNAME option. 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.

   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.

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 which 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. If the archive log data set is password-protected, be certain to use the PASSWORD option of the NEWLOG statement. Also, 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 which were created later than the BSDS copy.

c. Update passwords in the replacement BSDS.

   The BSDS contains passwords for the archive log data sets and for selected DB2 databases. To ensure that the passwords in the replacement BSDS reflect the current passwords used by your installation, use the change log inventory ARCHIVE and SYSTEMDB statements with the PASSWORD option.

d. 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.

e. Update the archive 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 the active log data set is password-protected, be certain to use the PASSWORD option.

   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. If the active log data set is password-protected, be certain to use the PASSWORD option.

f. 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 which 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. If the active log data sets are password-protected, be certain to use the PASSWORD option of the NEWLOG statement.

g. 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 off-loaded, 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; however, if the active log data sets are password-protected, be certain to use the PASSWORD option of 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 4-8. Recovery from BSDS or Log Failure During Restart” on page 4-221 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 4-232.
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 *MVS/ESA 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 *MVS/ESA 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.

Performing a cold start of DB2 can also result in a down-level condition.

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 TABLESPACE
RECOVER 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.

For index page sets, 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' SQL code 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 new 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.

  Attention

  Accepting a down-level data set or disabling down-level detection might well cause data inconsistencies. Problems with inconsistent data resulting from this action are the responsibility of the user.

For more information about using the utilities, see *Utility Guide and Reference* .

How to control down-level detection: Use the subsystem parameter DLDFREQ to control how often the level ID of a page set or partition is updated. DB2 accepts any value between 0 and 65535. For example, to update the level ID after every checkpoint that a page set is open for update activity, set DLDFREQ=1. To update the level ID after every fifth checkpoint that the page set is open for update activity, set DLDFREQ=5. DLDFREQ=5 is the default. To disable down-level detection, set DLDFREQ=0.

Consider the following when you choose a value for DLDFREQ:

- How often do you use backup and restore methods outside of DB2's control?

  If you rarely use such methods, you do not need to update the level ID often.

- How many page sets are open for update at the same time?

  If DB2 updates level IDs frequently, you have extra protection against down-level page sets. However, if the level IDs for many page sets must be set at every checkpoint, you might experience a performance degradation.

- How often does the subsystem take checkpoints?

  If the subsystem takes checkpoints frequently, the level ID can be set after a larger number of checkpoints.

To activate a new value of DLDFREQ, rerun job DSNTIJUZ and restart DB2. For more information about subsystem parameters, see Section 2 of *Installation Guide* .

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 TABLESPACE (*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 which 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 which, 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 TABLESPACE in Section 2 of *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:

       `DSNC510.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:

       `DSNC510.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.

   Note: 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

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 Section 3 of *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 " Section 2. Designing a Database" on page 2-1.

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. If you are recovering objects created in an ROSHARE OWNER database that are STOGROUP defined, and the physical VSAM data set does not exist, the command STOP DATABASE cannot be used. In this case, stop the DB2 subsystem.

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 for shared read-only data (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 4-6. Backing Up and Recovering Databases” on page 4-123.

Out of DASD 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 4-190
- “Procedure 2. Enlarge a Fully Extended Data Set (User-Managed)” on page 4-190
- “Procedure 3. Enlarge a Fully Extended Data Set (in a DB2 Storage Group)” on page 4-191
- “Procedure 4. Add a Data Set” on page 4-191
- “Procedure 5. Redefine a Partition” on page 4-191
- “Procedure 6. Enlarge a Fully Extended Data Set for the Work File Database” on page 4-192

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 4-192.

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."
- If the data set *has* reached the maximum number of VSAM extents, use either "Procedure 2. Enlarge a Fully Extended Data Set (User-Managed)" or "Procedure 3. Enlarge a Fully Extended Data Set (in a DB2 Storage Group)" on page 4-191, 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 4-191.
- 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 4-191.

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 5-100.

Procedure 2. Enlarge a Fully Extended Data Set (User-Managed)

1. For table spaces, be sure that you have a recent image copy to allow for recovery in case of failure during this procedure. Use the DSNUM option to identify the data set for table spaces.
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. For non-partitioned indexes on
   | a large partitioned table space the maximum is 128 data sets.
5. For a table space, use RECOVER to recover the data set that was redefined.
   # Identify the data set by the DSNUM option. RECOVER permits you to 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.

For an index, use the REBUILD INDEX utility to rebuild the index.

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 can give new values of PRIQTY and SECQTY in either the same or a new DB2 storage group.

2. Use one of the following utilities on the data set for the table space: REORG,
RECOVER or LOAD REPLACE. Use the REBUILD INDEX utility on the index. Keep in mind that no movement of data occurs until this step is completed.

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.

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. Use SQL to create a table (B) identical to the one needing enlargement (A).
2. Use SQL to copy the data to the newly-created table B.
3. Drop the table space containing the data. That also drops the table and any indexes, views, or synonyms dependent on it, and revokes all authorizations for the table and views that are dropped.
4. Redefine the table space, the partitioned index (with different key range values), the table (new A), and the nonclustering indexes.
5. Use SQL to copy the data of B to the new partitioned table A.
6. Re-create all the views and synonyms and re-grant authorization.

As an alternative to steps 2 and 5, you can do the following:

- Use SQL SELECT statements to unload the rows of the original table A into a file. Use the LOAD utility to load the rows from the file into the newly created table B.
- Use SQL SELECT statements to unload the rows of table B into a file. Use the LOAD utility to load the rows from the file into the new partitioned table A.

For large table spaces, this technique might be faster than copying data directly from one table to the other. For an example of using SQL to unload rows from a table into a file, see sample program DSNTIAUL.

Procedure 6. Enlarge a Fully Extended Data Set for the Work File Database

1. Use the command STOP DATABASE (DSNDB07) to ensure that no users are accessing the database.
2. Add space for extension to the DB2 storage group by one of these methods:
   - Use SQL to change the storage group, adding volumes as necessary.
   - Use SQL to create more table spaces in database DSNDB07.
3. Use the command START DATABASE (DSNDB07).

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 Section 2 of *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, -904 SQL return code (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.

For application directed access, a -30080 error status code is returned to the application. The SQLCA contains the VTAM diagnostic information which contains only the RCPRI and RCSEC 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 via 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

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) doesn't 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 (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, 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.

System Programmer Action: None.

Operator Action: Correct the condition described in the messages received at the console, restart TCP/IP.

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 4-70 and “Using VTAM Commands to Cancel Threads” on page 4-71.

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 via the DDF.

Symptom: Message DSNL500I is issued at the requester (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 via DDF.

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 SQL return code -904 (SQLSTATE '57011') with a reason code 00D3103D, indicating that a resource is unavailable. SQL return code -30080 is issued for applications using application directed-access.

System Programmer Action: Refer to the description of 00D3103D in Section 4 of *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 4-133. For data sharing, see Chapter 6 of *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. If an integrated catalog facility catalog does not already exist, run job DSNTIJCA to create a user catalog.
2. Use the access method services IMPORT command to import the integrated catalog facility catalog.

3. 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.

4. Use IDCAMS DELETE NOSCRATCH to delete all catalog and user objects. (Because step 2 imports a user ICF catalog, the catalog contains data sets that do not exist on DASD.) 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.

5. Recover the BSDS:

   a. Use the access method services REPRO command to restore the contents of the two BSDS data sets (allocated in the previous step). The most recent BSDS image will be found in the first file (file number one) on the latest archive log tape.

   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

      For 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 since 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. Group buffer pool checkpoint information is stored in the BSDS and needs to be included from the most recent archive log.

   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.

      If you are using dual BSDSs, make sure both of them are included in the jobs.

   e. If you are using the DB2 distributed data facility, run the change log inventory utility with the DDF statement to update the LOCATION, LUNAME, and PASSWORD 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).

6. Optionally, you can restore archive logs to DASD. Archive logs are typically stored on tape, but restoring them to DASD 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.

7. 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 Section 3 of *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 4-203.

8. Modify DSNZP*xxx* to defer processing of all databases:

   a. Run the DSNTINST CLIST in UPDATE mode. See Section 2 of *Installation Guide* .

   b. From panel DSNTIPB select “Databases to Start Automatically.” 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. From panel DSNTIPB select “Operator Functions.” You are presented with panel DSNTIPO. Type RECOVERYSITE in the SITE TYPE field. Press Enter to continue.

   d. 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.

9. 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. 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.
- 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.

DB2 discards any log information in the bootstrap dataset 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.

10. Enter the command START DB2 ACCESS(MAINT).

Data Sharing

Start one DB2 with ACCESS(MAINT). DB2 will prompt you to start each additional DB2 subsystem in the group.

Even though DB2 marks all table spaces for deferred restart, log records are written so that inabort and inflight units of recovery are backed out. Incommit 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 which 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 Since 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.

11. 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 4-113. From an install SYSADM authorization ID, enter the RECOVER INDOUBT command for all affected transactions.

    If you attempt this from an MVS console, you will receive messages resulting from an attempt to do authorization checking when no tables exist yet.

12. To recover the catalog and directory, follow these instructions:

    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 of the disaster. 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.

       See What to Do About Utilities in Progress on page 4-203 for information on how to recovery catalog and directory table spaces on which utilities were running.

d. Use the command TERM UTILITY to terminate any utilities in progress on catalog or directory table spaces.

e. Recover the rest of the catalog and directory objects starting with DBD01 in the order shown in Section 2 of *Utility Guide and Reference*under RECOVER utility topic "Recovering Catalog and Directory Objects".

13. Use any method desired to verify the integrity of the DB2 catalog and directory. Migration step 1 in Section 2 of *Installation Guide* lists one option for verification. The catalog queries in member DSNTESQ of data set DSN510.SDSNSAMP can be used after the work file database is defined and initialized.

14. Define and intialize 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.

15. If you use the distributed data facility, recover the objects in the communications data base.

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 Section 2 of *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. See What to Do About Utilities in Progress on page 4-203 for information on how to recover table spaces on which utilities were running. You cannot restart a utility at the recovery site that was interrupted a the disaster site. Use the TERM command to terminate any utilities running against user table spaces.

    a. Issue the SQL query

       ```
       SELECT * FROM SYSIBM.SYSTABLEPART WHERE STORTYPE='E';
       ```

       to determine which, if any, of your table spaces are user-managed. To allocate user-managed table spaces, use the access method services DEFINE CLUSTER command.

    b. If your user table 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.

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.

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) since FORCE resets any page set exception conditions described in “Database Page Set Control Records” on page X-86.

For the following utility jobs, take 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 57, and take the specified actions.

Table 57 (Page 1 of 2). Actions to Take 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.
	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 *copy spec*	If the reload phase completed, then the table space is complete after you recover it to the current time. Recover the indexes.
	If the reload phase did not complete, then recover the table space to a prior point in time. Recover the indexes.

Table 57 (Page 2 of 2). Actions to Take when LOAD is Interrupted

LOAD options specified	What to do
LOG NO *copy spec* SORTKEYS	If the build phase completed, then recover to the current time, and recover the indexes. If the build 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.

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 58, and take the specified actions.

Table 58. Actions to Take 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. 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. 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 *copy spec*	If the reload phase completed, then the table space is complete after you recover it to the current time. Recover the indexes. 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 *copy spec* SORTKEYS	If the build phase completed, then recover to the current time, and recover the indexes. 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.
SHRLEVEL CHANGE	If the switch phase completed, terminate the utility. Recover the table space to the current time. Recover the indexes. 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 can not use online REORG. For those table space that can use online REORG find the options you specified in Table 58, and take 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 TABLESPACE on the catalog and directory in the order shown in Section 2 of *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 ctalog and directory objects, the RECOVER to current stops at the state before the REORG began. 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 all indexes.

If you have image copies from immediately before REORG failed, run RECOVER TABLESPACE with the option TOCOPY to recover the catalog and directory, in the order shown in Section 2 of *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 4-197. Many steps are similar to a regular disaster recovery, so we don't go into detail on those steps.

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 don't have to constantly ship image copies, the takeover time for 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 4-207
- "Maintaining the Tracker Site" on page 4-210
- "The Distaster Happens: Making the Tracker Site the Takeover Site" on page 4-210

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 tracker site DB2 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.
- Downlevel 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, restoring DB2 libraries, and tailoring installation job DSNTIJIN to create DB2 catalog data sets.
2. Modify the subsystem parameters as follows:
   - Set the TRKSITE subsystem parameter to YES.
   - Optionally, set the SITETYP parameter to RECOVERYSITE fi the full image copies this site will be receiving are created as remote site copies.
3. Use the access method services command DEFINE CLUSTER 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 don't require indexes, you don't have to rebuild them for recovery cycles.

4. Send full image copies of all the primary site's DB2 data the tracker site.

Attention: 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 4-209 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 4-197.

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=nnnnnnnnnnn,FORWARD=NO,BACKOUT=NO
```

where *nnnnnnnnnnn* 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. Wtih 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 up-to-date. See “Media Failures during LOGONLY Recovery” on page 4-210 for information about what to do if a media failure occurs during LOGONLY recovery.

   a. Recover the catalog and directory

      See *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 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.

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.

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 completeing 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 in sync with the SYSUTILX at the primary site, the tracker cycle updates must be discarded before the next tracker cycle.

There are two ways to play for recovering SYSUTILX:

- Send a full image copy fo DSNDB01.SYSUTILX for each recovery cycle. At the tracker site, you would use normal recovery (that is fullimage copy and logs).
- Use DSN1COPY copies of SYSUTILX at the tracker site and LOGONLY recoveries. tHe sequence of steps is:

  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 *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 bad and you are using DB2 storage groups, you cannot use the ALTER STOGROUP statement to remove the bad volume and add another as is documented for a non-tracker system. Instead, you must remove the bad 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 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 Distaster Happens: Making the Tracker Site the Takeover Site

If a disaster occurs at the primary site, the tracker site must become the "takeover" site.

1. Restore the BSDS and register the archive log from the last archive you received from the primary site.
2. **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 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). Then run your recovery jobs as you did during recovery cycles: that is, using LOGONLY recovery when possible.

- If you did receive 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 4-207 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, tecover 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 4-203 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. Don't forget to rebuild indexes, including user-defined indexes on the DB2 catalog.

Resolving Indoubt Threads

This section describes problem scenarios involving indoubt threads resulting from the following types of error conditions:

All scenarios are developed in the context presented in "Description of the Environment" on page 4-212. 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 86 on page 4-213. 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**.

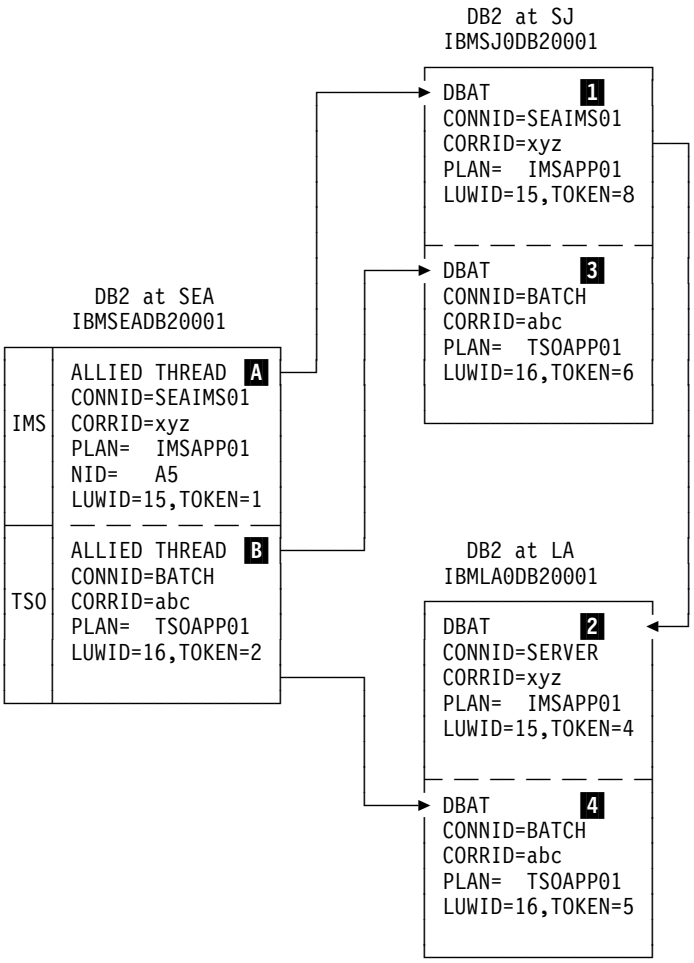


Figure 86. Resolving Indoubt Threads. Results of issuing -DIS THD TYPE(ACTIVE) at each DB2 system.

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 86. 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 reconnection and automatic resolution of 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 which 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 Resulting in 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 4-161.

System Action: This is the same process as described in “Resolution of Indoubt Units of Recovery” on page 4-161.

System Programmer Action: This is the same process as described in “Resolution of Indoubt Units of Recovery” on page 4-161.

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 4-161.

DB2 Outage at Application Requestor Resulting in 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 via 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 may have more accessible functions to determine the decision. Even though the SEA DB2 cold started, you may be able to determine the decision from its recovery log. Or, if the failure occurred before the decision was recorded, you may 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 which 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 an 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 which IMS is trying to resolve with DB2. The message indicates that DB2 has no knowledge of the thread which 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 reconnection with 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 Application Server Resulting in DB2 Cold Start

Problem: This problem is similar to “DB2 Outage at Application Requestor Resulting in DB2 Cold Start” on page 4-216. 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, since it would complete the two-phase commit process with the SEA DB2.

As described in “Communication Failure Between Two Systems” on page 4-213, 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 4-213. 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 4-8. 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 4-171
"Archive Log Failure" on page 4-175
"BSDS Failure" on page 4-177.

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 Section 3 of *Messages and Codes* indicate that restart failed because of some problem not related to a log error, refer to Section 3 of *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 4-131. Methods of recovery are described under "Unresolvable BSDS or Log Data Set Problem during Restart" on page 4-242.

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 6 of *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	00D10264	00D10267	00D1032A	00D1032C
00D10262	00D10265	00D10268	00D1032B	00E80084
00D10263	00D10266	00D10329		

Figure 87 on page 4-222 illustrates the general problem:

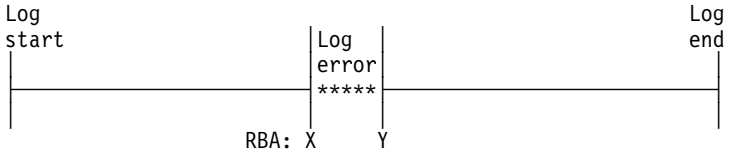


Figure 87. General Problem of Damaged DB2 Log Information

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 87 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 Section 3 of *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 4-248.

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 4-223
- “Failure during Forward Log Recovery” on page 4-233
- “Failure during Backward Log Recovery” on page 4-238.

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 4-223.

DSNJ100I	“Unresolvable BSDS or Log Data Set Problem during Restart” on page 4-242.
DSNJ107I	“Unresolvable BSDS or Log Data Set Problem during Restart” on page 4-242.
DSNJ1191	“Unresolvable BSDS or Log Data Set Problem during Restart” on page 4-242.
DSNR002I	None. Normal restart processing can be expected.
DSNR004I	“Failure during Forward Log Recovery” on page 4-233.
DSNR005I	“Failure during Backward Log Recovery” on page 4-238.
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 4-244) 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 4-248) 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 4-242, it is not necessary to use Resolving Inconsistencies Resulting from Conditional Restart.

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 *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 4-242.
- 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 4-226. 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 88 illustrates the problem on the log.

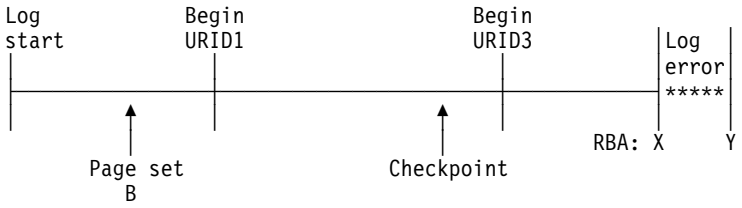


Figure 88. Failure during Log Initialization

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 59 on page 4-226.

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” on page 4-226.

Description of Failure during Current Status Rebuild

Figure 89 illustrates the problem on the log.

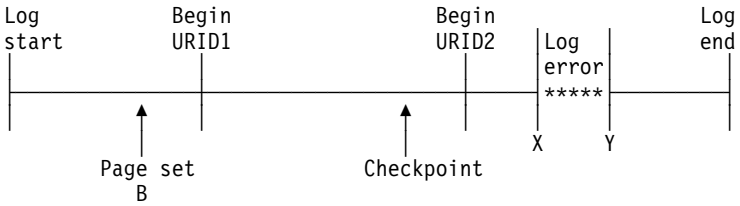


Figure 89. Failure during Current Status Rebuild

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 59 on page 4-226.

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” on page 4-226.

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 88 on page 4-224 and in Figure 89 on page 4-225. 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 59. Use that procedure to find X and Y.

Table 59. Abend Reason Codes and Messages

Abend Reason Code	Message	Procedure Name and Page	General Error Description
00D10261	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10262	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10263	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10264	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10265	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10266	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10267	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10268	DSNJ012I	RBA 1, page 4-226	Log record is logically damaged
00D10329	DSNJ106I	RBA 2, page 4-227	I/O error occurred while log record was being read
00D1032A	DSNJ113E	RBA 3, page 4-227	Log RBA could not be found in BSDS
00D1032B	DSNJ103I	RBA 4, page 4-227	Allocation error occurred for an archive log data set
00D1032B	DSNJ007I	RBA 5, page 4-228	The operator canceled a request for archive mount
00D1032C	DSNJ104I	RBA 4, page 4-227	Open error occurred for an archive and active log data set
00E80084	DSNJ103I	RBA 4, page 4-227	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 167 on page X-87 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 4KB boundary (X'7429000').

Continue with step 2 on page 4-229.

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 167 on page X-87 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 4KB boundary (X'7429000').

Continue with step 2 on page 4-229.

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 Section 3 of *Utility Guide and Reference*.

Figure 167 on page X-87 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 4-244.

If there is a value for X, continue with step 2 on page 4-229.

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 Section 3 of *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 Section 3 of *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 4-229.

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 Section 3 of *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 4-229.

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 4-244 or the procedure under “Unresolvable BSDS or Log Data Set Problem during Restart” on page 4-242.

      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 4-244 or the procedure under “Unresolvable BSDS or Log Data Set Problem during Restart” on page 4-242.

      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 can 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 can be possible to accurately identify the page sets that were made inconsistent. To make the identification, extracting a summary of the log activity up to the point of damage in the log by using the DSN1LOGP utility described in Section 3 of *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 90 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 90. 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 91 on page 4-231. 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 91. 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 DASD. 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, since 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 4-101.
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 4-248.

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 *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 4-242.

- 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 4-235. Continue reading this chapter to obtain a better idea of what the problem is.

Figure 92 illustrates the problem on the log.

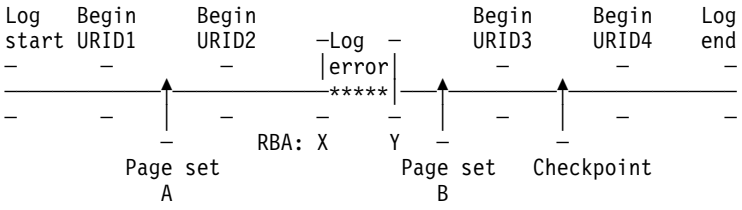


Figure 92. Illustration of Failure during Forward Log Recovery

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 92. 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 DASD, see “Tuning Database Buffer Pools” on page 5-49.

Step 1: Find the Log RBA after the Inaccessible Part of the Log

The log damage is shown in Figure 92 on page 4-234. 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 60. Use that procedure to find X and Y.

Table 60. Abend Reason Codes and Messages

Abend Reason Code	Message	Procedure Name and Page	General Error Description
00D10261	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10262	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10263	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10264	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10265	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10266	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10267	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10268	DSNJ012I	RBA 1, page 4-235	Log record is logically damaged
00D10329	DSNJ106I	RBA 2, page 4-236	I/O error occurred while log record was being read
00D1032A	DSNJ113E	RBA 3, page 4-236	Log RBA could not be found in BSDS
00D1032B	DSNJ103I	RBA 4, page 4-237	Allocation error occurred for an archive log data set
00D1032B	DSNJ007I	RBA 5, page 4-237	The operator canceled a request for archive mount
00D1032C	DSNJ104E	RBA 4, page 4-237	Open error occurred for an archive log data set
00E80084	DSNJ103I	RBA 4, page 4-237	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 167 on page X-87 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 4-237.

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 167 on page X-87 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 Section 3 of *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 4-237.

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' isn't 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 Section 3 of *Utility Guide and Reference*.

Figure 167 on page X-87 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 4-237.

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 *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 Section 2 of *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 4-237.

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 Section 3 of *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 4-237.

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 90 on page 4-230 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 91 on page 4-231; and for an example of the program that results in the output, see Figure 90 on page 4-230.

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 4-248 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 *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 4-242.
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.” Continue reading this chapter to obtain a better idea of how to fix the problem.

Figure 93 illustrates the problem on the log.

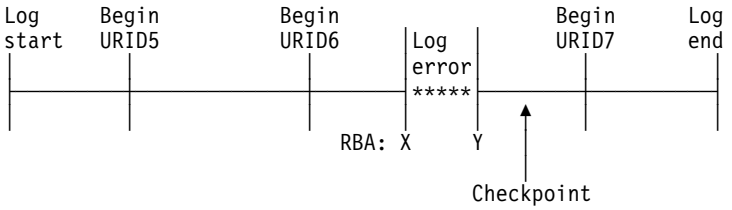


Figure 93. Illustration of Failure during Backward Log Recovery

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 DASD. A subsequent restart of DB2 causes these changes to be written to DASD 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 91 on page 4-231 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 91 on page 4-231 resulted from the invocation shown in Figure 90 on page 4-230. 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 91 on page 4-231.

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 91 on page 4-231.

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 4-248 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 Section 2 of *Installation Guide*.

For instructions about adding an old archive data set refer to “Changing the BSDS Log Inventory” on page 4-94. Also see Section 3 of *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 4-131.

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 4-244.

Preparing for Recovery of Restart

See “Preparing to Recover to a Prior Point of Consistency” on page 4-131 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 Section 2 of *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 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 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 shut-down 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” on page 4-245 or “Excessive Loss of Data in the Active Log” on page 4-246.

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 4-179.
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 4-105.

   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 4-248.

Excessive Loss of Data in the Active Log

By studying "Total Loss of Log" on page 4-245, 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 4-245. *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 useable.
3. Use access method services DELETE to delete all active log data sets which are no longer useable.
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 useable 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 which 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 Section 3 of *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 4-248.

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. 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 which 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 which 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 4-183 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 Section 2 of *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 4-144 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.” 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” 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 4-144, 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 Section 7 of *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 Section 7 of *Diagnosis Guide and Reference*. Look for these pointers and manually ensure their consistency.

  Hash and link pointers exist only in the DB2 directory 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 Section 2 of *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.

Section 5. Performance Monitoring and Tuning

Chapter 5-1. Planning Your Performance Strategy

The first step toward improving performance is planning. This chapter describes:

- As an overview of the rest, "Further Topics in Monitoring and Tuning"

As elements of planning a strategy:

- "Managing Performance in General" on page 5-8
- "Establishing Performance Objectives" on page 5-9
- "Planning for Monitoring" on page 5-13
- "Reviewing Performance Data" on page 5-15
- "Tuning DB2" on page 5-17

And as a summary for users of previous versions of DB2 for MVS/ESA:

- "Enhancements in DB2 Version 5" on page 5-17

Information on performance monitoring and tuning in a data sharing environment is presented in *Data Sharing: Planning and Administration*.

Further Topics in Monitoring and Tuning

"Chapter 5-2. Analyzing Performance Data" on page 5-25 is a general guide to analyzing and investigating performance issues.

"Chapter 5-3. Improving Response Time and Throughput" on page 5-37 deals with space allocation, buffer pool and data set usage, processor resources, and how response time is reported.

"Chapter 5-4. Tuning DB2 Buffer, EDM, RID, and Sort Pools" on page 5-49, has recommendations for monitoring and tuning those objects.

"Chapter 5-5. Improving Resource Utilization" on page 5-73, deals with managing data sets, logging, DASD, main storage, the resource limit facility, and performance options in MVS.

"Chapter 5-6. Managing DB2 Threads" on page 5-115, deals with DB2 threads and work load management.

"Chapter 5-7. Improving Concurrency" on page 5-137, deals with concurrency and locking.

"Chapter 5-8. Tuning Your Queries" on page 5-203, deals with writing queries that are as efficient as possible.

"Chapter 5-9. Maintaining Statistics in the Catalog" on page 5-243, deals with catalog statistics, when to reorganize, and when to rebind.

"Chapter 5-10. Using EXPLAIN to Improve SQL Performance" on page 5-261, deals with the principal means of monitoring access path selection and improving the performance of your SQL, especially queries.

"Chapter 5-11. Monitoring and Tuning in a Distributed Environment" on page 5-315, deals with performance in a distributed environment and in particular with stored procedures.

Throughout this section, bear in mind the following:

- Performance objectives that can be reasonably measured by tools now available are emphasized. 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. There is no guarantee that this book provides the best or most appropriate advice for any specific site. In particular, the advice in this section approaches situations from a performance viewpoint only; at some sites, there can be other factors of higher priority that make some recommendations in this section inappropriate.
- The recommendations are general. Actual performance statistics are not included because such measurements are highly dependent on work load 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.

   If performance is thoroughly satisfactory, consider whether to:

   - Monitor less, because monitoring itself uses resources; or
   - Continue monitoring to generate a history of performance to compare with future results.

   If performance has not been satisfactory, then:
5. Determine the major constraints in the system.
6. 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.
7. Tune your system by adjusting its characteristics to improve performance.
8. Return to step 3 above and continue to monitor the system.

Periodically, or after significant changes to your system or work load, 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 completed within a given time)
- System availability, including mean time to failure and the durations of down times

Objectives such as those define the work load 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 work load, 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 transaction 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 are under 2 seconds, or that the average response time will 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 your transactions.

Defining the Work Load

To define the work load of the system, begin by assigning transactions to types. For each type, describe a preliminary work load profile that includes:

- A definition of the transaction type in terms of its function and its volume. You are likely to have many transactions that perform the same general function (for example, order entry) and have an identifiable work load profile. There are also diverse transactions, such as SPUFI or QMF queries. For a transaction that is already processed by DB2, you can get a summary of work load 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, DASD 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.

ForTransactions:

- Availability of 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 transactions per minute, hour, day, or week
- Number of I/O operations per user transaction (DASD and terminals)
- Average and maximum processor usage per transaction, 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:

- Windows for data reorganization, utilities, data definition activities, and BIND processing
- Batch processing load
- Length of batch window
- Number of transactions (records to process, data reorganization, 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 work load. Changes in design at this stage, before there is 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 transaction type, convert the estimates of resource requirements into measurable objectives. Include statements about the transaction 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 work loads. 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 transactions that have exceptionally heavy requirements for processing or database access, or establish individual targets for those transactions.

- 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 will slow response further.

- DASD response time. I/O operations are generally responsible for much of the internal processing time of a transaction. Consider all I/O operations that affect a transaction.

- Existing work load. Consider the effects of additional work load 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 work load. 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 DASD subsystem work load based on transaction dialogs.
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 work load against the actual work load.
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 temporary 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 transactions or combination of transactions that cause 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 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 G. Using Tools to Monitor Performance” on page X-173. 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 will provide for four levels of monitoring: continuous, periodic, detailed, and exception. These levels are discussed in the sections that follow. “A Monitoring Strategy” on page 5-15 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. The overhead can be significant, however; for details, see “Minimize the Use of DB2 Traces” on page 5-44.

With CICS, there is less need to run with accounting class 2. Application and non-DB2 processing take place under the CICS main TCB. SQL activity takes place under the SQL TCB, so 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 work load 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 5-44 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 Section 4 of *Application Programming and SQL Guide*; for information about database design, see “ Section 2. Designing a Database” on page 2-1.

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 G. Using Tools to Monitor Performance” on page X-173 suggests various techniques and methods.

If you have access path problems, refer to “Chapter 5-10. Using EXPLAIN to Improve SQL Performance” on page 5-261 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 “Concurrency Scenario” on page 5-193.

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 5-44 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, transaction 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 work load, 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 transaction 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 X-179.

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 5-110 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 can happen when programs have to open data sets.

See “Monitoring System Resources” on page X-175, “Using MVS, CICS, and IMS Tools” on page X-175, and “Statistics Trace” on page X-178.

To what degree was DASD 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 X-175 and “Statistics Trace” on page X-178.

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 X-175.

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 X-178.

Do any figures indicate design, coding, or operational errors?

1. Are DASD, I/O, log, or processor resources heavily used? If so, was that expected at design time? If not, can the heavy use be explained in terms of heavier use of transactions?
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 DASD, channel, or path problems?
6. Are there any abends or dumps?

See “Monitoring System Resources” on page X-175, “Statistics Trace” on page X-178, and “Accounting Trace” on page X-179.

Were there any bottlenecks?

1. Were any critical thresholds reached?
2. Are any resources approaching high utilization?

See “Monitoring System Resources” on page X-175, and “Accounting Trace” on page X-179.

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 work load?

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 work load. 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 Section 2 of *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.

Enhancements in DB2 Version 5

These changes in DB2 Version 5 affect throughput, concurrency, resource usage, and performance.

Sysplex Query Parallelism: DB2 Version 5 expands parallel processing power for queries across the parallel sysplex. Installations using data sharing can take advantage of this query enhancement. For information about query parallelism in general, see “Parallel Operations and Query Performance” on page 5-299.

Information specifically related to Sysplex query parallelism is in Chapter 7 of *Data Sharing: Planning and Administration*.

Stored Procedures Enhancements:

- A stored procedure can return multiple result sets.

  A stored procedure running at a DB2 for OS/390 server can issue multiple SELECT statements on behalf of a remote client. The result sets can be returned using a single network message, thus greatly reducing the elapsed time and processor cost of query result sets. See *Application Programming and SQL Guide* for more information about this feature.

- You can set dispatching priority for stored procedures instead of having them all run under the priority of the stored procedures address space. See "Using Workload Manager to Set Performance Objectives" on page 5-124 for more information.

- Stored procedures can run as subprograms, which take less time for initialization and termination.

- The COMMIT_ON_RETURN column of the SYSIBM.SYSPROCEDURES catalog table indicates that DB2 issues an implicit COMMIT on behalf of the stored procedure upon return from the CALL statement. This can reduce the length of time locks are held and can reduce network traffic, especially when the request is from an application using DRDA level 1. See Section 6 of *Application Programming and SQL Guide* for more information.

Better Performance for Applications that use DRDA: Network processing for distributed requests is one of the biggest contributors to overall application processing cost. The following changes reduce that cost:

- The ability to cache authorizations for packages, described on 5-21

- Fewer messages exchanged for dynamic SQL

  A single message exchange on the network is used to both send and receive a dynamic SQL query. For example, you could invoke a DB2 SELECT statement dynamically as shown below.

  ```
  DECLARE CURSOR C1 FOR S1;
   PREPARE S1 INTO :sqlda FROM :query;
    OPEN C1;
    FETCH C1;

   ⋮
  ```

  With previous releases of DB2, the dynamic SQL SELECT statement requires two network message exchanges. Using Version 5, the dynamic SQL SELECT statement requires just a single network message exchange. Statements that do not use parameter markers will see this benefit. See *Application Programming and SQL Guide* for more information.

- Reduced processing cost for VTAM messages

  When DB2 runs with VTAM Version 4 Release 4, DB2 is able to reduce its processing overhead. VTAM processing costs are reduced by approximately 15 percent at the DB2 server and 10 percent at the DB2 requester. You will also see improved performance when a DB2 server transmits reply messages, because of enhancements in VTAM that reduce the number of times data is moved during network I/O operations.

- Faster block fetching

  Queries with large numbers of columns and rows will exhibit the greatest performance improvements. Improvements will be seen in the user's accounting trace data.

- The clause OPTIMIZE FOR *n* ROWS available for DRDA applications

  As was previously available for local applications and for those using DB2 private protocol, you can limit the number of rows returned in each block by using the OPTIMIZE FOR *n* ROWS clause on the SELECT statement.

Quicker Exchange of Data between ASCII Client and DB2 Server: You can now store tables as ASCII on DB2 for OS/390. This means that for data that needs to be ordered in ASCII sequence, you don't have to use a field procedure to convert the data to ASCII and sequence it properly.

DEFER(PREPARE) Available for Packages: The ability to defer statement preparation has been extended to packages (previously, it was just for plans). See *Application Programming and SQL Guide* for more information about using deferred PREPARE.

Caching Prepared Statements: A new caching feature improves dynamic query performance by eliminating most of the cost of duplicate prepares for the same SQL statement. Prepared SQL statements are cached so subsequent PREPARE requests for the same SQL statement avoid the optimization process. Because the cache is global to the DB2 subsystem, different application processes can share prepared statements. For more information about caching prepared statements, see Section 6 of *Application Programming and SQL Guide*.

Save Prepared Statements across Commits: At bind time, you can specify the KEEPDYNAMIC bind option which indicates that prepared SQL statements should be preserved past a commit point. This eliminates the need to repeat the PREPARE statement in your application. For more information, see Chapter 2 of *Command Reference*.

Reoptimize at Run Time: At bind time, the real value of host variables, parameter markers, or special registers is not known, so a default factor is used to make access path decisions. In some cases, the default produces an access path that results in unacceptable performance. In DB2 Version 5, you can choose a new BIND option, REOPT(VARS), to have DB2 reconsider the access path at run time when the values are known. See “Using REOPT(VARS) to Change the Access Path at Run Time” on page 5-224 for more information.

Option for Index-only Retrieval of Varying-length Data: Previously, DB2 could not retrieve varying-length data without going to the data page. Now, you can specify YES on a new subsystem parameter, RETVLCFK, and DB2 can return data to your application using only the index. When RETVLCFK is set to YES, data is returned from the index and is padded with blanks to the maximum length of the column. If your application is sensitive to these blanks, keep the default value, NO.

If you choose YES, you must rebind plans and packages to enable the change.

Improvements for Queries That Use Correlated Columns: You can collect more statistics with DB2 to improve access path selection for indexes that have

correlated columns, such as CITY and STATE. For more information, see Section 2 of *Utility Guide and Reference*.

CASE Expressions: CASE expressions can reduce the elapsed time of queries that temporarily put data in "buckets" based on multiple returned values. See Section 2 of *Application Programming and SQL Guide* for more information.

Improvements for Non-column Expressions: Predicates of the form "column op non_col_expression" can now be evaluated sooner. Predicates such as "DATE_COL < CURRENT DATE - 50 DAYS" can take advantage of the improvement. See "Summary of Predicate Processing" on page 5-210 for more information.

Index Access for IN Noncorrelated Subquery Predicates: DB2 can use an index to access predicates with noncorrelated IN subqueries. For example, in the following statement, DB2 uses TAB1's index on PROG for matching index access for TAB1.

```
UPDATE TAB1
SET SDATE = ?, STIME = ?
 WHERE PROG IN
          (SELECT MASTER
           FROM TAB2
           WHERE INCLUDE = ?)
```

For another example, assume that a clustering, nonunique index exists on PRODUCT_NBR. The index can be used for matching index access for the predicate PRODUCT_NBR IN( ).

```
SELECT ....
FROM PRODUCT
WHERE   PRODUCT_NBR IN
       (SELECT S.PRODUCT_NBR
        FROM SEARCH_TOKENS S
        WHERE S.SEARCH_TOKEN LIKE 'SHOE%')
```

Smarter Use of Implicitly Equal Unique Indexes: DB2 is better at recognizing when an index is unique. Assume the following:

Unique Index IX1: COL1, COL2
Index IX2: COL1, COL2, COL3, COL4, COL5

In this situation, IX2 must also be unique because it contains a subset of columns that are defined as unique. However, before Version 5, DB2 did not recognize that situation in statements like the following:

```
SELECT COL1, COL2, COL3, COL4, COL5
  FROM TABLE1
 WHERE COL1 = 'A'
   AND COL2 = 'B';
```

Before Version 5 DB2 would choose Index IX1 because it returns one row using a fully matching unique index. However, Index IX2 is a better choice because it also returns only one row, *and* it provides index-only access. DB2 does not have to go to the data page at all to return the row.

Better estimates for filtering: DB2 improves its estimates of filtering for non-Boolean term predicates using AND, such as the following:

```
SELECT *
 FROM   TAB
 WHERE (C1 = A AND C2 = B) OR ( C3 = C AND C4 = D).....
```

This better filtering estimate also applies to predicates that use LIKE and a pattern-matching character, such as:

```
WHERE NAME LIKE '%TIE%'
```

DB2 Visual Explain: DB2 Version 5 includes a Visual Explain tool that uses the PLAN_TABLE to obtain information to display an access path in graphical or tabular format from a workstation client. The Visual Explain tool is invoked from a workstation to give you:

- A display of a selected access path
- A detailed description of the access path
- Recommendations on access paths that might influence DB2 to change the access path after the recommendations are applied and the statements are rebound
- The ability to invoke EXPLAIN for dynamic SQL statements
- The ability to provide statistics for referenced objects of an access path

A subsystem parameter browser is included with the Visual Explain tool. This workstation package enhances your ability to optimize SQL performance and provides a visual analysis of the performance of DB2 applications. For information on using the DB2 Visual Explain tool, which is a separately packaged CD-ROM provided with your DB2 license, see *DB2 Visual Explain online help*.

EXPLAIN Information Available in Trace: You can obtain more information about the access path chosen at run time with IFCID 0022, so you can better analyze what changed in the access path from bind time to run time.

Caching Authorizations for Packages: Caching information about package authorization avoids repeated checks on the catalog at run time for successive uses of the same package by the same user. See " Section 3. Security and Auditing" on page 3-1 for more information.

Locking and Concurrency Enhancements:

- The LOCKPART option of CREATE and ALTER TABLESPACE lets you specify that only partitions that are accessed are locked. This option is useful for data sharing, in those cases in which you are routing work to specific members of the data sharing group to create affinity between a specific partition and a member, or when you run batch jobs on separate partitions to avoid locking conflicts. See Chapter 7 of *Data Sharing: Planning and Administration* for more information.

  For table spaces defined with LOCKPART YES, you can also the PART option of LOCK TABLE to lock individual partitions. This can help reduce the number of locks that are acquired for applications that access different partitions without affecting concurrency.

- The MAXROWS option of CREATE and ALTER TABLESPACE is an easier way to get the concurrency of row locking with the lower cost of page locks. Previously, you had to do this by using page locking and by padding rows to ensure that only one row would be placed on a page. Now, by specifying MAXROWS 1, you can tell DB2 to always keep one row on a page.
- The PIECESIZE option of CREATE and ALTER INDEX gives you a way to reduce physical contention on nonpartitioned indexes. See Chapter 6 of for more information about the PIECESIZE option.
- The MEMBER CLUSTER option of CREATE TABLESPACE gives you a way to reduce page P-lock contention on space map pages for applications that do heavy sequential insert processing from different members of a data sharing group. See Chapter 7 of for more information about MEMBER CLUSTER.
- You can use a new 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 the U or S lock on all the qualified pages or rows.

  Here is an example:

  ```
  SELECT ...
   FOR UPDATE OF WITH RS KEEP UPDATE LOCKS;
  ```

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

  For 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.

More Partitions and Larger Tables: In Version 5, one table space can hold a terabyte of data, increased from 64 gigabytes and can have up to 254 partitions of 4 gigabytes each, increased from 64 partitions of 1 gigabyte each.

By distributing data among more partitions, you can shorten processing time for utilities that can run on separate partitions of data. Very large table spaces also give your database the flexibility to grow without having to separate the data into separate table spaces.

More Rows on a Page: The maximum number of rows per page is changed from 127 to 255 for all table spaces except those for the catalog and directory. This increase is available automatically without having to alter the table space for compression first. For existing table spaces, the change takes effect on a page-by-page basis when new data is inserted or deleted. Use REORG to change an entire table space.

Greater Availability with REORG: REORG now gives you increased access to your data or indexes during REORG, including read/write access during most phases. Note however that increased access to data might cause the REORG utility to take longer to complete. For more information about new REORG options, see Section 2 of *Utility Guide and Reference*.

| ***Faster LOAD and REORG:*** In addition to internal enhancements that mean reduced processor time, you can specify the new option SORTKEYS for LOAD and REORG to avoid writing index keys to work files. By not writing to an intermediate work data set, elapsed time is reduced. There is a new COPYDDN statement for both utilities that allows you to produce an image copy as part of running LOAD or REORG. This makes data available sooner by avoiding the separate copy step. See Section 2 of *Utility Guide and Reference* for more information about these options.

| ***Reduced Processing time for RECOVER INDEX:*** Changes to RECOVER INDEX save processing time in almost all cases. The one exception is for segmented table spaces when a list of indexes is specified. The ability to specify the piece size for nonpartitioned indexes can also help reduce the elapsed time needed to recover those indexes.

| ***Faster RUNSTATS using Sampling:*** A new sampling option improves performance by reducing the processing required to collect nonindexed column statistics. For example, you can define a percentage of the table rows, such as 10 percent, to be used in calculating nonindexed column statistics.

| In general, you can expect the following reduction in processing time:

| `(100-`*`samplepercent`*`) × 0.5 = percent reduction`

| For example, if you choose 50% sampling, the processing cost can be reduced up to 25 percent. See Section 2 of *Utility Guide and Reference* for more information.

| ***Striping for BSAM Data:*** DB2 online utilities can take advantage of extended sequential dataset support in DFSMS/MVS by using “striping” for BSAM data set access. BSAM striping transfers data between DASD and memory at a faster rate than an individual DASD can handle. BSAM striping performs best for large, physical sequential data sets with large block sizes and high sequential processing activity. This function requires certain specific hardware and software prerequisites. See *Release Guide* for more information.

| ***New Utility to Preformat Active Log Data Sets:*** DB2 provides a new stand-alone utility, DSNJLOGF, that allows you to preformat active log data sets before bringing them online to DB2. This can avoid delays when writes must occur to new active log data sets. For more information, see Section 3 of *Utility Guide and Reference*.

| ***New Option to Preformat Data Sets*** For applications that have heavy insert activity into new table spaces, you can preformat pages during LOAD or REORG instead of during insert processing. After data has been loaded or reorganized, DB2 preformats the remaining pages up to the high allocated RBA in the table space or partition, and in the associated index spaces. If you can preallocate a table space before it is used, use PREFORMAT when preformatting is causing measurable delays in your insert processing, or if you must have predictable elapsed times for insert processing. For more information about the PREFORMAT option, see Section 2 of *Utility Guide and Reference*.

New DESCRIBE INPUT Statement for Better Performance DESCRIBE INPUT obtains information about the input parameter markers of a prepared statement. This support improves performance for dynamic SQL applications and many ODBC applications by reducing the number of network messages that need to be exchanged when an application is executing dynamic SQL with input host variables

and does not know the correct data type of the input host variables ahead of time.
By using DESCRIBE INPUT, you can ask the DBMS to describe what an SQL
statement looks like and avoid the expense of catalog lookups for determining input
parameter marker data.

Chapter 5-2. Analyzing Performance Data

This chapter presents:

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 5-26
3. A suggested procedure for analyzing problems within DB2, in “A General Approach to Problem Analysis in DB2” on page 5-32

Investigating the Problem Overall

When analyzing performance data, keep in mind that almost all symptoms of poor performance are magnified when there is contention. If there is a slowdown in DASD, for example:

- Transactions can pile up, waiting for data set activity
- Transactions can wait for I/O and locks
- Paging can be delayed

In addition, there are more transactions in the system, and therefore 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, DASD 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 enqueue the same item. 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 transactions. 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 5-309.

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 X-107 for information about the format of data from DB2 traces.

You can also use the tools for performance measurement described in “Appendix G. Using Tools to Monitor Performance” on page X-173 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.

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.

Major Items in the Report: Figure 94 on page 5-28 shows a sample DB2 PM accounting report, short layout. It contains selected fields from the accounting record and gives overall totals.

The GRAND TOTAL shows all the packages and DBRMs executed for all the accounting records included in this report.

Near the end of the report is a listing of the plans with the highest values for:

- Processing (TCB) time spent in DB2 (see **A** in Figure 94 on page 5-28)
- Waiting time spent in DB2 (see **B** in Figure 94 on page 5-28)

Those values were derived using the TOP List function of DB2 PM.

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 95 on page 5-29.

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 95 on page 5-29.)

Major Items on the Report

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 IMSPARS.

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

```
   LOCATION: SYS1DSN2                        DB2 PERFORMANCE MONITOR (V5)                          PAGE: 1-1
      GROUP: DSN2                               ACCOUNTING REPORT - SHORT                    REQUESTED FROM: NOT SPECIFIED
     MEMBER: SE11                                                                                      TO: NOT SPECIFIED
  SUBSYSTEM: SE11                                     ORDER: PLANNAME                        INTERVAL FROM: 08/14/96 11:41:10.15
DB2 VERSION: V5                                       SCOPE: MEMBER                                  TO: 08/14/96 11:41:10.15

                          #OCCURS #ROLLBK SELECTS INSERTS UPDATES DELETES CLASS1 EL.TIME CLASS2 EL.TIME GETPAGES SYN.READ LOCK SUS
PLANNAME                  #DISTRS #COMMIT FETCHES   OPENS  CLOSES PREPARE CLASS1 CPUTIME CLASS2 CPUTIME BUF.UPDT TOT.PREF #LOCKOUT
------------------------- ------- ------- ------- ------- ------- ------- -------------- -------------- -------- -------- --------

PLAN1                        115       8    0.97    1.91   22.65    1.89       7.486930       6.070091  1585.19    15.10     9.22
                               0     107   29.61    1.97    0.00    0.00       0.868575       0.689688    29.65    61.66        0

    ---------------------------------------------------------------------------------------------------
    |PROGRAM NAME        TYPE      #OCCURS  SQLSTMT  CL7 ELAP.TIME   CL7 CPU TIME  CL8 SUSP.TIME  CL8 SUSP|
    |PU12301             DBRM            1    10.00       0.842631       0.076351       0.759055      7.00|
    |PU12302             DBRM            4    14.00       2.728067       0.139169       2.545810     13.75|
    |PU12304             DBRM            5    59.20       2.399819       0.773367       1.118798     50.80|
    |PU12305             DBRM            4    69.00       6.716764       0.718430       5.779797     92.50|
    |PU12306             DBRM            4    37.50       3.729345       0.765479       2.482065     70.25|

⋮

⋮
    |PU12316             DBRM            2    31.00       3.194021       0.646627       2.367049    121.50|
    ---------------------------------------------------------------------------------------------------

PLAN2                       1116       9    0.99    0.99    6.04    0.99       3.206735       4.511614    23.51     0.08     7.04
                               0    1107    8.03    1.99    0.00    0.00       0.146241       0.092230    15.79     0.72        2

    ---------------------------------------------------------------------------------------------------
    |PROGRAM NAME        TYPE      #OCCURS  SQLSTMT  CL7 ELAP.TIME   CL7 CPU TIME  CL8 SUSP.TIME  CL8 SUSP|
    |PU22301             DBRM           45     0.00       0.650992       0.000000       0.000000      0.00|
    |PU22302             DBRM           47     0.00       0.155940       0.000000       0.000000      0.00|

⋮

⋮

***  GRAND TOTAL  ***
                            3165      17    1.36    0.42    2.95    0.42       2.885499       3.787906    71.44     1.02     3.64
                               0    3148    7.48    1.63    0.00    0.00       0.146102       0.087293     6.68     2.76        4

    ---------------------------------------------------------------------------------------------------
    |PROGRAM NAME        TYPE      #OCCURS  SQLSTMT  CL7 ELAP.TIME   CL7 CPU TIME  CL8 SUSP.TIME  CL8 SUSP|
    |ALL PROGRAMS        BOTH         2563    10.42       0.690285       0.064153       0.578860      3.95|
    ---------------------------------------------------------------------------------------------------

TOP FIELD: PROCESSING (TCB) TIME SPENT IN DB2 A                  TOP NUMBER REQUESTED:    4

       PLANNAME                                                        VALUE    PAGE
----   ------------------------------------------------------   ---------------  ----------
   1   PLAN1                                                        0.689688    1-1
   2   PLAN3                                                        0.067977    1-2
   3   PLAN4                                                        0.045680    1-2
   4   PLAN2                                                        0.004205    1-1

TOP FIELD: WAITING TIME SPENT IN DB2 B                           TOP NUMBER REQUESTED:    4

       PLANNAME                                                        VALUE    PAGE
----   ------------------------------------------------------   ---------------  ----------
   1   PLAN1                                                        5.380403    1-1
   2   PLAN3                                                        4.578372    1-2
   3   PLAN2                                                        4.419383    1-1
   4   PLAN4                                                        2.331325    1-2

ACCOUNTING REPORT COMPLETE
```

Figure 94. Partial Accounting Report, Short Layout

IMS time, check the report from EPDM, IMSPARS, 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

```
   LOCATION: SYS1DSN2                    DB2 PERFORMANCE MONITOR (V5)                    PAGE: 1-1
      GROUP: DSN2                           ACCOUNTING REPORT - LONG                REQUESTED FROM: NOT SPECIFIED
     MEMBER: SE11                                                                            TO: NOT SPECIFIED
  SUBSYSTEM: SE11                                 ORDER: PLANNAME                   INTERVAL FROM: 08/14/96 11:41:10.15
DB2 VERSION: V5                                   SCOPE: MEMBER                             TO: 08/14/96 11:41:10.15

PLANNAME: PU22301

 AVERAGE       APPL (CLASS 1)   DB2 (CLASS 2)   IFI (CLASS 5)    CLASS 3 SUSP.   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
 ------------  --------------   --------------  --------------   --------------  ------------  --------    --------------------------
                                                                        C
 ELAPSED TIME        5.773449         3.619543             N/P   LOCK/LATCH  1.000181              1.09    #OCCURRENCES    :     80
  CPU TIME           0.141721         0.093469             N/P   SYNCHRON. I/O  A    0.002096       0.13    #ALLIEDS        :     80
  TCB                0.048918   B     0.004176             N/P   OTHER READ I/O  D   0.000000       0.00    #ALLIEDS DISTRIB:      0
  TCB-STPROC         0.092802         0.089294             N/A   OTHER WRTE I/O  E   0.000000       0.00    #DBATS          :      0
  PAR.TASKS          0.000000         0.000000             N/A   SER.TASK SWTCH  F   0.860814       1.04    #DBATS DISTRIB. :      0
 SUSPEND TIME             N/A         2.832920             N/A   ARC.LOG(QUIES)  G   0.000000       0.00    #NO PROGRAM DATA:      0
  TCB                     N/A         2.832920             N/A   ARC.LOG READ  H     0.000000       0.00    #NORMAL TERMINAT:     80
  PAR.TASKS               N/A         0.000000             N/A   DRAIN LOCK  I       0.000000       0.00    #ABNORMAL TERMIN:      0
 NOT ACCOUNT. L           N/A         2.847060             N/A   CLAIM RELEASE  J    0.000000       0.00    #CP/X PARALLEL. :      0
 DB2 ENT/EXIT             N/A             8.96             N/A   PAGE LATCH  K       0.000000       0.00    #IO PARALLELISM :      0
 EN/EX-STPROC             N/A            41.74             N/A   STORED PROC.        0.629187       0.04    #INCREMENT. BIND:      0
 DCAPT.DESCR.             N/A              N/A             N/P   NOTIFY MSGS.        0.000000       0.00    #COMMITS        :     80
 LOG EXTRACT.             N/A              N/A             N/P   GLOBAL CONT.        0.340642       7.37    #ROLLBACKS      :      0
                                                                TOTAL CLASS 3       2.832920       9.67    UPDATE/COMMIT   :   8.66

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  INDEX            0       0       0   ESCAL.(SHARED)      0.00         0
DELETE        1.00        80    SET CURR.SQLID         0  TABLESPACE       0       0       0   ESCAL.(EXCLUS)      0.00         0
                                SET HOST VAR.          0  DATABASE         0       0       0   MAX LOCKS HELD      8.47        15
DESCRIBE      0.00         0    SET CUR.DEGREE         0  STOGROUP         0       0       0   LOCK REQUEST       31.74      2539
DESC.TBL      0.00         0    SET RULES              0  SYNONYM          0       0     N/A   UNLOCK REQUEST      2.13       170
PREPARE       0.00         0    CONNECT TYPE 1         0  VIEW             0       0     N/A   QUERY REQUEST       0.00         0
OPEN          2.00       160    CONNECT TYPE 2         0  ALIAS            0       0     N/A   CHANGE REQUEST     10.46       837
FETCH         8.66       693    SET CONNECTION         0  PACKAGE        N/A       0     N/A   OTHER REQUEST       0.00         0
CLOSE         0.00         0    RELEASE                0                                      LOCK SUSPENS.       0.31        25
                                CALL                   0  TOTAL            0       0       0   LATCH SUSPENS.      0.15        12
                                ASSOC LOCATORS         0                                      OTHER SUSPENS.      0.00         0
DML-ALL      20.32      1626    ALLOC CURSOR           0  COMMENT ON       0                  TOTAL SUSPENS.      0.46        37
                                DCL-ALL               80  LABEL ON         0

⋮
```

Figure 95. Partial Accounting Report, Long Layout

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 (TWAIT=YES) or was diverted to the pool (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. If it is lengthy, the problem is in the application program, CICS, IMS, or the overall system.

Synchronous I/O Suspension Time: This is the total application wait time for synchronous I/Os. In the DB2 PM accounting report, check the number reported for SYNCHRON. I/O (**A**).

If the number of read 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 5-10. Using EXPLAIN to Improve SQL Performance” on page 5-261.

- Changes in the application. Check the “SQL ACTIVITY” section and compare with previous data. There might have been some inserts which 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. We recommend running 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 5-63.
- A RID pool failure. Refer to “Increasing RID Pool Size” on page 5-69.
- A system-wide problem in the EDM pool. Refer to “Tuning the EDM Pool” on page 5-66.

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 5-63.
- I/O contention. In general, each synchronous read I/O typically takes from 15 to 25 milliseconds, depending on the DASD 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 5-93.

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 TCB TIME (**B**).

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 “Concurrency Scenario” on page 5-193.

In the DB2 PM accounting report, see the field LOCK/LATCH (**C**).

Asynchronous 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 1.7 to 2.2 milliseconds per page. For list prefetch, the rule of thumb is 3 to 4 milliseconds per page.

In the DB2 PM accounting report, asynchronous read suspensions are reported in the field OTHER READ I/O (**D**).

Asynchronous 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 3 to 4 milliseconds per page.

In the DB2 PM accounting report, asynchronous 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:

- Log write I/Os for commit and abort processing
- The opening and closing of data sets
- Updating the SYSLGRNG directory table
- Preformatting DASD
- The extension of data sets

In the DB2 PM accounting report, this information is reported in the field SER.TASK SWTCH (**F**). 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 5-163, 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 5-163, 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 (as do most utilities) instead of a page lock in order 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**).

Other DB2 Time: The DB2 accounting class 2 elapsed time that is not recorded as class 2 TCB time or class 3 suspensions. The most common contributors to this category are:

- MVS paging
- Processor wait time

In the DB2 PM accounting report, this information is reported in the field NOT ACCOUNT. (**L**).

Comparing Elapsed Times from the Report

To analyze a response time problem you can isolate data from the DB2 PM accounting report to compare the breakdown of elapsed times. This method assumes that you are not analyzing operations that were performed in parallel. See "Using DB2 Trace" on page 5-310 for more information about accounting reports for parallel operations.

Table 61 illustrates the differences between the various accounting times reported in Figure 95 on page 5-29.

Table 61. A Comparison of DB2 Accounting Times

	Application Times (Class 1)	In DB2 Times (Class 2)	Not in DB2 Times
Processor Resource Consumption	TCB 0.048918	TCB 0.004176	Class 1 TCB Time minus Class 2 TCB Time
Wait Time (DB2 measured)	Class 3 2.832920	Class 3 2.832920	Not Applicable
Other Time	Class 1 elapsed minus Class 1 TCB minus Class 3 2.891611	Class 2 elapsed minus Class 2 TCB minus Class 3 0.782447	Application and transaction manager wait time

Notes:

Application Times = In-DB2 Times + Not-in-DB2 Times
Elapsed Times = TCB Time + Wait Time + Other Time
In-DB2 Elapsed Time = Class 2 TCB Time + Class 3 + Other Class 2 Time
Wait time = Class 1 elapsed – Class 2 elapsed + Class 1 TCB – Class 2 TCB

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 TCB time[9], class 3 time, and "other" time. If your performance monitoring tool does not specify times other than Class 2 and Class 3, then you can determine "other" time with the following formula:

```
Other time = Class 2 elapsed time - Class 2 TCB time - Total class 3 time
```

2. If the class 2 TCB 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 TCB 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 5-49.
   - 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 TCB time for packages to determine which package or DBRM has the largest TCB time or the greatest increase in TCB 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 5-10. Using EXPLAIN to Improve SQL Performance" on page 5-261 for help in understanding access path selection and analyzing access path problems. See *DB2 Visual Explain online help* and the DB2 Visual Explain that is packaged separately with your DB2 license for a workstation display of your EXPLAIN output.
   - Check the counts in the locking section of the DB2 PM accounting report. If locking activity has increased, see "Chapter 5-7. Improving Concurrency"

[9] The majority of processor time is captured in the TCB time. When evaluating processor resource consumption under the allied thread, use the TCB time as the indicator of processor utilization. The SRB time is collected across the entire allied address space, and therefore it cannot be fully attributed to DB2. Because the SRB time cannot always be interpreted consistently and the actual SRB time is relatively small compared to the TCB time, you can ignore the SRB time.

on page 5-137. 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 95 on page 5-29).
   - If LOCK/LATCH (**C**), DRAIN LOCK (**I**), or CLAIM RELEASE (**J**) time is high, see "Chapter 5-7. Improving Concurrency" on page 5-137.
   - If SYNCHRON. I/O (**A**) time is high, see 5-29.
   - 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:
     – Sort work files
     – Tables in which rows are inserted beyond the formatted range
     – Active log data sets (can occur when new active log added or by frequent use of the command ARCHIVE LOG)

     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 "other" time is high, check for paging activity and excessive processor waits.
   - 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 96 on page 5-35 shows which reports you might use, depending on the nature of the problem, and the order in which to look at them.

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 96 on page 5-35:

- 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 "Concurrency Scenario" on page 5-193.
- Follow the third branch for a **Global problem**. For example, an excessive average elapsed time per I/O. There could be a wide variety of transactions that suffer similar problems.

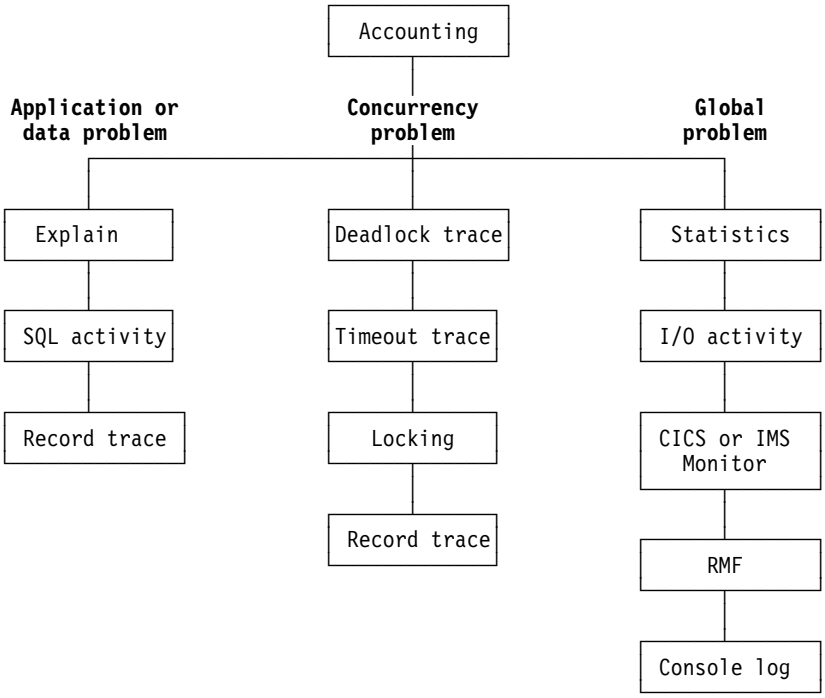


Figure 96. DB2 PM Reports Used for Problem Analysis

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 DSNRCT 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 5-3. Improving Response Time and Throughput

Response time consists of the following three components:

- Processor resource consumption, which is shown on Figure 95 on page 5-29 as “TCB 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 5-40
- “Reducing the Amount of Processor Resources Consumed” on page 5-43

The following sections describe ways to accomplish these goals. The chapter concludes with an overview of how various DB2 response times are reported.

Data Sharing: 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 *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 Data Access Statistics Current”
- “Reserve Free Space in Table Spaces and Indexes” on page 5-38
- “Make Buffer Pools Large Enough for the Work Load” on page 5-40
- “Ensure Allocation in Cylinders” on page 5-40

Using indexes can also minimize I/O operations. For information on indexes and access path selection see “Overview of Index Access” on page 5-276.

Use RUNSTATS to Keep Data Access 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 “Using RUNSTATS to Monitor and Update Statistics” on page 5-249 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

General-use Programming Interface

Reserve Free Space in Table Spaces and Indexes

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.

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:

Faster insertion of rows
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 DASD occupied
Less information transferred per I/O
More pages to scan
Possibly more index levels
Less efficient use of buffer pools and 3990 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 CREATE or ALTER TABLESPACE statement.

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 Section 2 of *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. 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 segment size.

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 DASD space, because it might not be cost-justified. Use of PCTFREE or FREEPAGE depends on the ratio of insertions to deletions, and the distribution of that activity across the index or table space.

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 index is a type 2 index and 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.

- Rows are larger than half a page, because you cannot insert a second row on a page.

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.

Hints: Consider setting PCTFREE to 0 to save storage space if:

- Inserts are in ascending order by the key of the clustering index
- Inserts, and updates that lengthen the row, are few.

For concurrency, use 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.

For **read-only** tables or indexes, use PCTFREE 0 and FREEPAGE 0.

End of General-use Programming Interface

Make Buffer Pools Large Enough for the Work Load

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 5-57 for more information.

Ensure Allocation 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 RECOVER INDEX 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.

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*.

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
- “Recommendations for Data Set Distribution” on page 5-41
- “Ensure Sufficient Primary Allocation Quantity” on page 5-42
- “Parallel Operations and Query Performance” on page 5-299.

For information on I/O scheduling priority, see “MVS Performance Options for DB2” on page 5-108.

Create Additional Work File Table Spaces

If your applications require any of the following, allocate additional work file table spaces on separate DASD 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)
- Temporary tables

The recommendation for work file DASD volumes is to have whichever is more:

- Five
- One-fifth the maximum number of data partitions

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.

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 5-71. See also Figure 38 on page 2-71 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 DSNC510
      DSETPASS DBADMIN;
   ```

4. Enter the following command:

   ```
   -START DATABASE (DSNDB07)
   ```

Recommendations for Data Set Distribution

Avoid I/O contention and increase throughput through the I/O subsystem by placing frequently used data sets on fast DASD and by distributing I/O activity wisely.

Use Partitioned Table Spaces

Assign the most frequently used data sets to the faster DASDs at your disposal. One way to do this is by creating a partitioned table space, allowing you to split a table into different data sets and to place the frequently used partition on a fast device. If performance analysis shows excessive I/O contention for a nonpartitioned table space, you can partition the tables and spread them around to different volumes. Placing frequently used data sets on fast DASD devices also improves performance for nonpartitioned table spaces.

Distribute the I/O

Frequently used data sets or partitions should be allocated across your available DASDs so that I/O operations are distributed. You should also 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. Make sure that device and control unit utilization is efficient. Put the more heavily accessed data sets on faster, less utilized devices. This helps reduce I/O contention, and takes more advantage of the benefits of parallel processing.

General-use Programming Interface

| ***Spread Data Sets of Nonpartitioning Indexes:*** For nonpartitioning indexes, 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. By making this a small value, for example, you can end up with many more data sets than you would by using the default of 2 gigabytes (4 gigabytes for table spaces defined as LARGE). 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 five datasets for the nonpartitioned index and your nonpartitioned index is 10MB, specify PIECESIZE 2M. If your nonpartitioned index is likely to grow, choose a larger value. Remember that 32 pieces is the limit if the underlying tablespace is not defined as LARGE and that 128 is the limit if the underlying tablespace is defined as LARGE.

Keep your PIECESIZE value in mind when you are choosing values for primary and secondary quantities. Ideally, the value of your primary quantity plus the secondary quantities should be evenly divisible into PIECESIZE to avoid wasting space.

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 DASD.

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 2-125.

Specifying 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 that evenly divides into PIECESIZE. For more information about PIECESIZE, see Chapter 6 of *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

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 Section 2 of *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 5-128 for details.
- If you are using the Recoverable Resource Manager Services attachment facility, see Section 6 of *Application Programming and SQL Guide* for more information about reusing threads.

Reduce the Number of CICS Threads per Region

Avoid unnecessary CICS threads. A CICS thread is anchored in a TCB, and unnecessary TCBs cause extra MVS dispatching overhead and inefficient use of processor cache. To avoid creating a large number of TCBs when the CICS attachment starts, do not specify a large number of protected threads (THRDS on the RCT). When the attachment starts, it creates a TCB for each protected thread.

To determine the optimum number of threads for your work load, you must do some experimentation. Much depends on the type of work load and the software release levels you have. For example, with CICS Version 4 and MVS 5.1 and later releases, you can probably have a larger number of threads than with earlier releases.

For more information about tuning for CICS, see “CICS Design Options” on page 5-128. See also *CICS/ESA Performance Guide*.

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

The DB2 accounting class 1 and 3 traces and statistics class 1, 3, 4, and 5 traces, together, cost only about 2% to 5% of processing overhead, in general. We recommend that you enable this collection of performance information, because it is necessary for capacity planning. Activate the statistics information by specifying YES for the fields SMF STATISTICS of installation panel DSNTIPN. For SMF ACCOUNTING, you must explicitly specify 1,3 to activate both classes.

Attention: Do not run accounting class 3 by default if you have a significant amount of latch contention in your system. If the number of latch contentions, as reported by statistics, is close to 1000 per second, the overhead of class 3 accounting is significant.

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 your usage of DB2 is very light, you might consider using the MVS SMF 89 | records for measured usage pricing. There is added overhead with this trace; if you | don't need it, don't turn it on. See the MVS SMF documentation for more information. The SMF type 89 record counts DB2 processor time for the system services, database services, and distributed data facility address spaces; the DB2 online utility, DSN1COPY; and cross-memory access to DB2. The record is included as part of an MVS trace that is enabled at the MVS system level.

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, though not as significantly as suppressing the performance trace.

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 DASD. If you must use varying-length columns, there are two considerations: retrieval performance and update performance. For the best retrieval performance, place varying-length columns at the end of a row. For the best update performance, place the columns to be updated at the end of a row. If you use both both retrieval and update operations, place the columns to be updated at the end, followed by the read-only varying-length columns.

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.

Considerations for Rebinding Certain Plans and Packages

SQL queries in applications that are bound in DB2 Version 3 or later releases, automatically undergo an internal optimization that could improve their performance, depending on the number of rows fetched. The SQL statements most likely to show a decrease in elapsed time are those that select a large number of columns and fetch hundreds of rows.

This optimization can be bypassed if service applied to DB2 has invalidated it. In this case, you can rebind the appropriate plans to once again take advantage of the internal optimization. To determine if the optimization has been invalidated by DB2 maintenance, check the BYPASS COL field in the Miscellaneous section of the DB2 PM statistics report (field QISTCOLS in IFCID 0002). If this field contains anything but zero, get the names of the plans or packages from IFCID 0224.

To start a trace for IFCID 0224, you must use one of the installation-defined trace classes (30-32):

```
-START TRACE (PERFM) CLASS(30) IFCID(224)
```

IFCID 0224 is written whenever the invalidated optimization is detected. The record contains the plan names or package names; rebind those plans or packages to pick up the optimization again.

How Response Time Is Reported

To correctly monitor response time, you must understand how it is reported. Response time can be measured in several different ways. Figure 97 on page 5-47 shows how some of the main measures relate to the flow of a transaction.

In Figure 97 on page 5-47, 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 5-309 for more information.

Elapsed times for stored procedures 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.

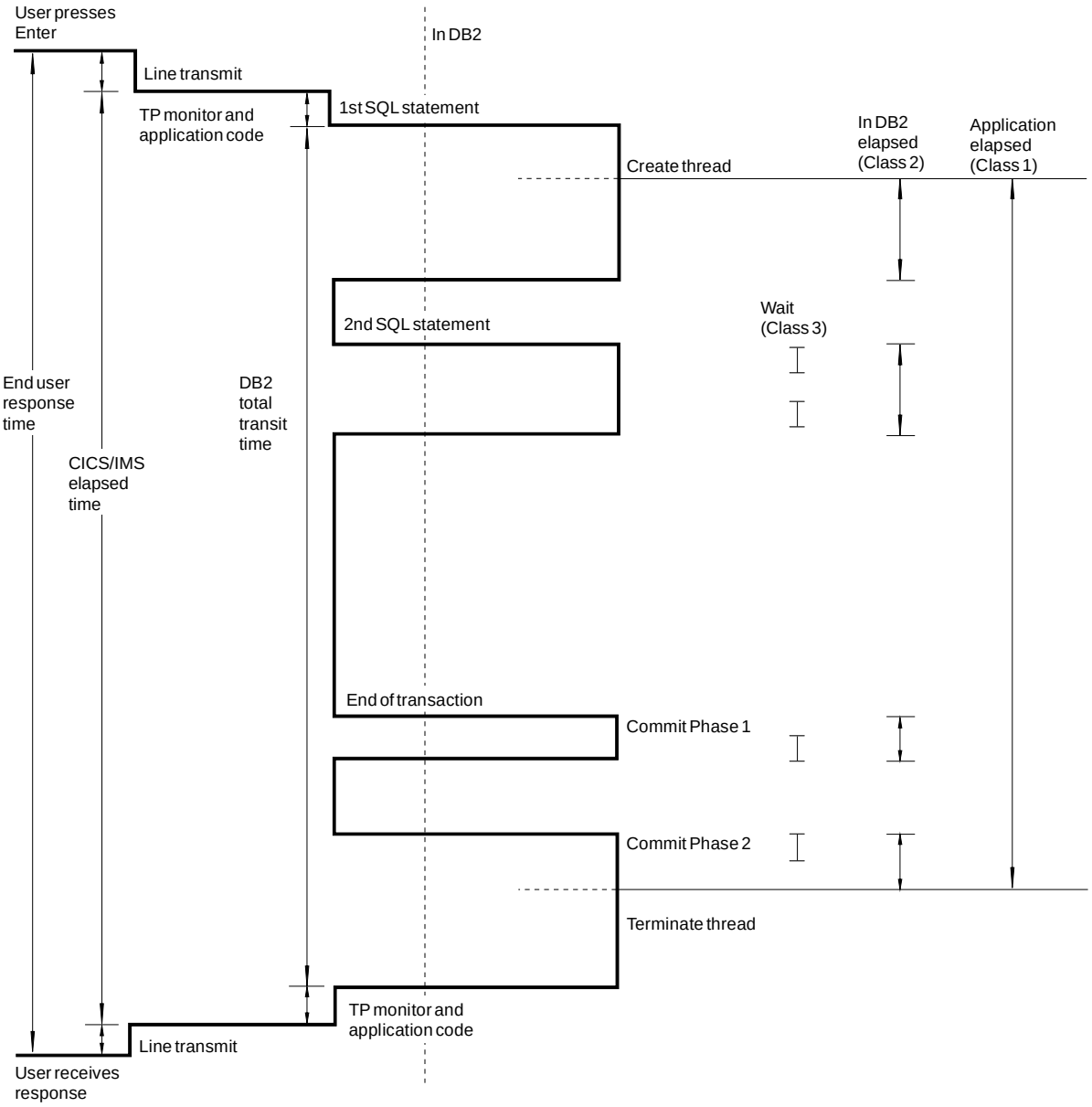


Figure 97. Transaction Response Times. Class 1 is standard accounting data. Class 2 is elapsed and processor time in DB2. Class 3 is elapsed wait time in DB2. Standard accounting data is provided in IFCID 0003, which is turned on with accounting class 1. When accounting classes 2 and 3 are turned on as well, IFCID 0003 contains additional information about DB2 times and wait times.

Chapter 5-4. 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 Data in a Table Space or Partition” on page 2-63 and “Compressing Your Data” on page 5-102. This chapter covers the following topics:

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 containing 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 DASD. Avoiding the need to retrieve data from DASD results in faster performance.

If the row is changed, the data in the buffer must be written back to DASD eventually. But that write operation might be delayed until DB2 takes a checkpoint, or until one of the related write thresholds is reached.

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 DASD I/O operation.

DB2 allows you to use up to 50 buffer pools that contain 4KB buffers and 10 buffer pools that contain 32KB 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.

Buffer Pools and Hiperpools

If your DB2 subsystem is running under MVS Version 4 Release 3 or later, and the Asynchronous Data Mover Facility of MVS is installed, you have the option of using hiperspaces to extend DB2's virtual buffer pools. A *hiperspace* is a storage space of up to 2GB that a program can use as a data buffer. Hiperspace is addressable in 4KB blocks; in other words, it is page addressable. For more information on hiperspace, see *MVS/ESA Programming: Extended Addressability Guide*.

DB2 cannot directly manipulate data that resides in hiperspace, but it can transfer the data from hiperspace into a regular DB2 buffer pool much faster than it could get it from DASD. To distinguish between hiperpools and buffer pools, we now refer to the regular DB2 buffer pools as virtual buffer pools.

On systems that have the prerequisite hardware and software, 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 1.6GB.
- The second level of storage, the hiperpool, uses the MVS/ESA hiperspace facility to utilize expanded storage only (ESO) hiperspace. The sum of all hiperpools cannot exceed 8GB. Hiperpools are optional.

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 DASD: all read and write operations to data in the page, and all DASD 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 DASD read operations are not required for accessing data that resides in hiperspace, response time is shorter than for DASD retrieval. Retrieving pages cached in hiperpools takes only microseconds, rather than the milliseconds needed for retrieving a page from DASD, which reduces transaction and query response time.

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 98 illustrates the relationship between a virtual buffer pool and its corresponding hiperpool.

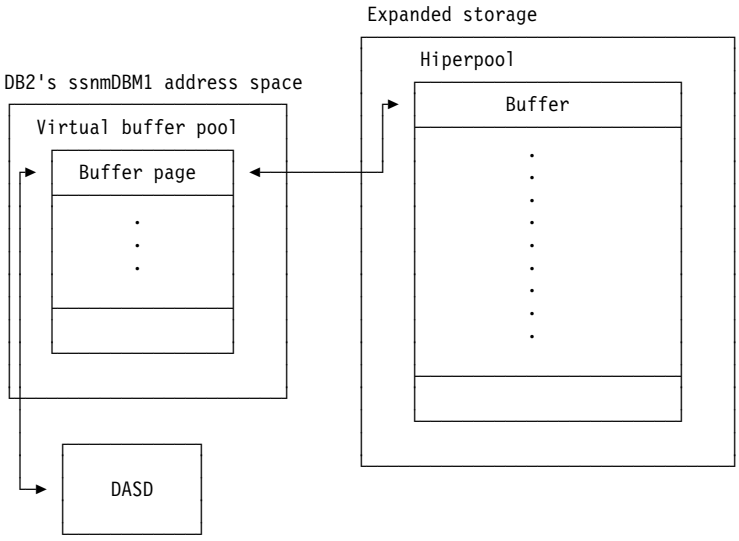


Figure 98. Relationship between Virtual Buffer Pool and Hiperpool

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. The maximum expanded storage available on ES/9000 processors is 8GB.

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.

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 DASD. After the updated page has been written to DASD, 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 5-291 for a complete description of sequential prefetch.

List Sequential Prefetch: This mechanism 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 Sequential Prefetch (PREFETCH=L)" on page 5-291.

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 5-53.

- 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 5-53.

Up to 32 4KB or 4 32KB pages can be written in a single I/O operation.

Installation Options

With DB2, you can use up to 60 different virtual buffer pools and hiperpools. For these virtual buffer pools and hiperpools, initially you specify the size and the CASTOUT attribute on panels DSNTIP1 and DSNTIP2. For information about those installation panels, see *Installation Guide*.

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 and DSNTIP2. You can modify the sizes of virtual buffer pools and hiperpools using the ALTER BUFFERPOOL command, so it is not important to choose an exact size initially.

The CASTOUT Attribute

Because expanded storage is a shared system resource, DB2 is not the only user of your MVS system's hiperspace. If DB2 monopolizes the available hiperspace, performance could be adversely affected. The CASTOUT attribute gives you some control over DB2's use of hiperspace.

You can define a hiperpool with a CASTOUT attribute of YES or NO. 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 can be recovered only when it is read again from DASD into the virtual buffer pool. 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 DASD.

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.

The initial setting for the CASTOUT attribute is not critical, but CASTOUT YES is recommended. You can change the CASTOUT attribute of a hiperpool with the ALTER BUFFERPOOL command.

Assigning a Table Space or Index to a Virtual Buffer Pool

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.

The table spaces and indexes of the directory (DSNDB01) and catalog (DSNDB06) are assigned to BP0; you cannot change that assignment. BP0 is also the default

virtual buffer pool for sorting. It has a default size of 2000 buffers, and a minimum of 56 buffers.

Buffer Pool Thresholds

The information under this heading, up to “Determining Size and Number of Buffer Pools” on page 5-57, is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page xi.

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 DASD when updates are completed.

Figure 99 shows the relationship between some of the virtual buffer pool thresholds and the updated, in-use, and available pages.

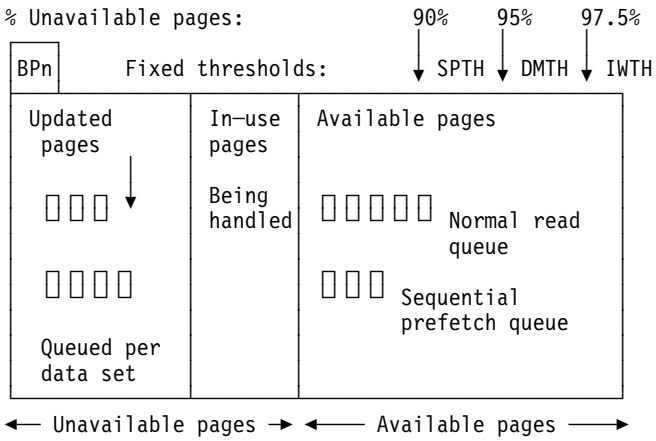


Figure 99. Database Virtual Buffer Pool. SPTH, DMTH, and IWTH are the performance critical thresholds.

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 or hiperpool, 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 DASD 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.

Variable Thresholds

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.

  Setting the threshold to 100% would allow 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 7 of *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.

- **Vertical Deferred Write Threshold (VDWQT)**

  This threshold is expressed as a percentage of the virtual buffer pool that might be occupied by updated pages from a single data set.

  The default value for this threshold is 10%. You can change that to any value from 0% to 90% by using the VDWQT keyword on the ALTER BUFFERPOOL command.

  This threshold is checked whenever an update to a page is completed. If the percentage of updated pages for the data set exceeds the threshold, writes are scheduled for that data set, up to 128 pages.

  Because any buffers that count toward VDWQT also count toward DWQT, setting VDWQT 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 VDWQT to a value greater than DWQT.

  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.

Guidelines for Setting Buffer Pool Thresholds

Because increasing DWQT and VDWQT allows updated pages to use a larger portion of the virtual buffer pool, setting DWQT and VDWQT to large values can have a significant effect on the other thresholds. For example, for a work load in which pages are frequently updated, and the set of pages updated exceeds the size of the virtual buffer pool, setting both DWQT and VDWQT to 90% would probably cause the sequential prefetch threshold (and possibly the data management threshold and the immediate write threshold) to be reached frequently.

If a virtual buffer pool is large enough, it is unlikely that the default values of either DWQT or VDWQT will ever be reached. In this case, there tend to be surges of write I/Os as deferred writes are triggered by DB2 checkpoints. Lowering the VDWQT and the DWQT could improve performance by distributing the write I/Os more evenly over time.

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.

Buffer Pools Used for Queries and Transactions: For a buffer pool used exclusively for query processing, it is reasonable to set VPSEQT and HPSEQT to 100%.

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.

Determining Size and Number of Buffer Pools

Considering the real storage and expanded storage that is available to DB2, it can help your applications and queries to make the virtual buffer pools large enough to increase the *buffer hit ratio*, which is a measure of how often a page access (a getpage) is satisfied without requiring an I/O operation.

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.

Calculating the Buffer Pool Hit Ratio

To determine the approximate buffer hit ratio, you first need to determine how many getpage operations did not require an I/O operation. To do this, subtract the number of pages read from DASD (both synchronously and using prefetch) from the total number of getpage operations. Then divide this number by the total number of getpage operations to determine the hit ratio.

For 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.

The highest possible value for the hit ratio is 1.0, which is achieved when every page requested is always in the buffer pool.

The lowest hit ratio is when the requested page is not in the buffer pool; in this case, the hit ratio is 0 or less. When the hit ratio is negative, it means that prefetch has brought pages into the buffer pool that are not subsequently referenced, either because the query stops before it reaches the end of the table space, or because

the prefetched pages are stolen by DB2 for reuse before the query can access them.

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 use a special type of getpage request that reserve an empty buffer without requiring that the page be read from DASD.

For work files, there is always a hit as DB2 reads the empty buffer for the input to sort, and then there is a read for the output. 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:

```
((getpages / 2) - I/O) / (getpages / 2)
```

Hit Ratio in Buffer Pool Statistics Report: The DB2 PM buffer pool statistics report also indicates the buffer pool hit ratio as shown in field **H** in Figure 102 on page 5-65.

Buffer Pool Size Guidelines

DB2 handles large virtual buffer pools very efficiently. Searching in large virtual buffer pools (100MB or more) does not use any more of the processor's resources than searching in smaller pools.

For processors dedicated to DB2, start with the default buffer pool sizes. You can increase the buffer pool size as long as the number of I/Os continues to decrease, or until paging becomes a problem. If your application uses random I/Os to access the data, the number of I/Os might not decrease significantly unless the buffer pool is larger than the table, and other applications require little concurrent buffer pool usage.

Problems with Paging: When the buffer pool size requirements are excessive (real storage plus expanded storage), the oldest buffer pool pages migrate to auxiliary paging storage. Subsequent access to these pages results in a page fault. I/O must bring the data into real storage. Paging of buffer pool storage impacts DB2 performance. The statistics for PAGE-INS REQUIRED FOR WRITE and PAGE-INS REQUIRED FOR READ shown in Figure 102 on page 5-65 are useful in determining if the buffer pool size setting is too large for available real storage.

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 DASD devices, 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 devices containing the work file table spaces is reduced.

Watch for Storage Paging: If the large buffer pool size results in excessive real storage paging to expanded storage, consider using hiperpools.

Choosing One or Many Buffer Pools

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 4KB buffer pool:

- Not enough total buffer space for more than 10 000 4KB buffers.
- No people 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 temporary tables use work files for those temporary 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.

Using the 32KB Buffer Pool

We recommend that you provide at least the default storage value for at least one 32KB buffer pool, even when all the tables use 4KB pages. This is because some SQL operations, such as joins, can create a result row that will not fit into a 4KB page.

Though the default storage value for at least one 32KB buffer pool is recommended, in general, the use of a 32KB buffer pool should be carefully considered. Data in table spaces that use a 32KB buffer pool is stored and allocated as 8 records, each 4KB in size. Inefficiencies can occur if small records are stored in table spaces that use a 32KB buffer pool. On the other hand, a 32KB page can be very good for predominately sequential processing where record size is greater than 1 KB. For example, only one 2100-byte record can be stored in a 4KB page, wasting almost half of the space, but storing the record in a 32KB page can significantly reduce this waste.

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 5-63, is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page xi.

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 size of a virtual buffer pool or hiperpool, some of the threshold values, or the hiperpool CASTOUT attribute for active or inactive virtual buffer pools or hiperpools.

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 100 on page 5-61. The operation captured by this report is the processing of sort work files for a query.

In Figure 100 on page 5-61, 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 between real storage and expanded storage.

  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 DASD.
- 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(20000) 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 101 on page 5-62.

```
+DISPLAY BPOOL(BP1) DETAIL
DSNB401I + BUFFERPOOL NAME BP1, BUFFERPOOL ID 1, USE COUNT 8
DSNB402I + VIRTUAL BUFFERPOOL SIZE = 20000 BUFFERS
             ALLOCATED       =    20000   TO BE DELETED   =        0
             IN-USE/UPDATED  =       11
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
             PARALLEL SEQUENTIAL   =  0   ASSISTING PARALLEL SEQT= 0
DSNB409I + INCREMENTAL STATISTICS SINCE 14:57:55 JAN 22, 1995
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 100. Sample Output from the DISPLAY BUFFERPOOL Command. This sample output shows buffer pool statistics for the processing of sort work files.

In Figure 101 on page 5-62, notice the following fields:

- You can verify the new hiperpool size by checking the HIPERPOOL SIZE field (**M**).
- In this example, the hiperpool size allocated (ALLOCATED **N**) is larger than the value for BACKED BY ES (**O**) 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 (**V**).

```
+DISPLAY BPOOL(BP1) DETAIL
DSNB401I + BUFFERPOOL NAME BP1, BUFFERPOOL ID 1, USE COUNT 8
DSNB402I + VIRTUAL BUFFERPOOL SIZE = 20000 BUFFERS
             ALLOCATED       =    20000   TO BE DELETED   =        0
             IN-USE/UPDATED  =       11
DSNB403I + HIPERPOOL SIZE M  = 20000 BUFFERS, CASTOUT = NO
             ALLOCATED N     =    20000   TO BE DELETED   =        0
             BACKED BY ES O  =    13929
DSNB404I + THRESHOLDS -
             VP SEQUENTIAL         = 80   HP SEQUENTIAL          = 80
             DEFERRED WRITE        = 50   VERTICAL DEFERRED WRT  = 10
             PARALLEL SEQUENTIAL   =  0   ASSISTING PARALLEL SEQT= 0
DSNB405I + HIPERSPACE NAME(S) - @011D31A
DSNB409I + INCREMENTAL STATISTICS SINCE 16:16:16 JAN 23, 1995
DSNB411I + RANDOM GETPAGE    =      156 SYNC READ I/O (R) =       11
           SEQ.   GETPAGE    =   132294 SYNC READ I/O (S) = P      0
           DMTH HIT          =        0
DSNB412I + SEQUENTIAL PREFETCH -
             REQUESTS        =     8253   PREFETCH I/O Q  =      103
             PAGES READ R    =      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 S =      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) -
                                                                    U
             HP READS T      =    35177   HP WRITES    =    35657
             READ FAILURES   =        0   WRITE FAILURES  = V      0
DSNB440I + PARALLEL ACTIVITY -
            PARALLEL REQUEST =        0  DEGRADED PARALLEL=        0

DSN9022I + DSNB1CMD '+DISPLAY BPOOL' NORMAL COMPLETION
```

Figure 101. Sample Output from the DISPLAY BUFFERPOOL Command. This output shows how the buffer pool statistics changed after the ALTER BUFFERPOOL command was issued.

- The value for SYNC READ I/O (**P**), which was 326 before the ALTER BUFFERPOOL command was issued, has decreased significantly.
- The values for PREFETCH I/O (**Q**) and PAGES READ(**R**) have decreased significantly because most of the requested pages are in the hiperpool, resulting in fewer pages that need to be fetched from DASD through sequential prefetch.
- SYNC HP READS (**S**) corresponds to the SYNC READ I/O (S) (**A**) value in Figure 100 on page 5-61.

- HP READS (**T**) shows the number of times data was read from the hiperpool into the virtual buffer pool.
- HP WRITES (**U**) 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 DASD.

For more information on the ALTER BUFFERPOOL or DISPLAY BUFFERPOOL commands, see Chapter 2 of *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 102 on page 5-65 shows

Increase the virtual buffer pool size or reduce the workload if:

- Sequential prefetch is inhibited. PREF.DISABLED-NO BUFFER (**A**) 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 (**B**)
  - PAGES WRITTEN PER WRITE I/O (**C**)

  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 DASD. 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 "Determining the Size of Active Logs" on page 5-96.

– *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 DASD. 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 (**D**) 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 CRITICAL THRESHOLD is zero.

- DM CRITICAL THRESHOLD (**E**) 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 (**F**)

  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.

TOT4K GENERAL	QUANTITY
CURRENT ACTIVE BUFFERS	1217.18
UNAVAIL.BUFFER-VPOOL FULL	0.00
NUMBER OF DATASET OPENS	1436.00
BUFFERS ALLOCATED - VPOOL	75500.00
BUFFERS ALLOCATED - HPOOL	55500.00
HPOOL BUFFERS BACKED	1972.91
DFHSM MIGRATED DATASET	0.00
DFHSM RECALL TIMEOUTS	0.00
HPOOL EXPANS. OR CONTRACT.	0.00
VPOOL EXPANS. OR CONTRACT.	0.00
VPOOL OR HPOOL EXP.FAILURE	0.00
CONCUR.PREF.I/O STREAMS-HWM	0.00
PREF.I/O STREAMS REDUCTION	0.00
PARALLEL QUERY REQUESTS	0.00
PARALL.QUERY REQ.REDUCTION	0.00
PREF.QUANT.REDUCED TO 1/2	0.00
PREF.QUANT.REDUCED TO 1/4	0.00

TOT4K READ OPERATIONS	QUANTITY
BPOOL HIT RATIO (%) **H**	73.12
GETPAGE REQUEST	1869.7K
GETPAGE REQUEST-SEQUENTIAL	1378.5K
GETPAGE REQUEST-RANDOM	491.2K
SYNCHRONOUS READS	54187.00
SYNCHRON. READS-SEQUENTIAL	35994.00
SYNCHRON. READS-RANDOM	18193.00
GETPAGE PER SYN.READ-RANDOM	27.00
SEQUENTIAL PREFETCH REQUEST	41800.00
SEQUENTIAL PREFETCH READS	14473.00
PAGES READ VIA SEQ.PREFETCH	444.0K
S.PRF.PAGES READ/S.PRF.READ	30.68
LIST PREFETCH REQUESTS	9046.00
LIST PREFETCH READS	2263.00
PAGES READ VIA LIST PREFTCH	3046.00
L.PRF.PAGES READ/L.PRF.READ	1.35
DYNAMIC PREFETCH REQUESTED	6680.00
DYNAMIC PREFETCH READS	142.00
PAGES READ VIA DYN.PREFETCH	1333.00
D.PRF.PAGES READ/D.PRF.READ	9.39
PREF.DISABLED-NO BUFFER **A**	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

TOT4K WRITE OPERATIONS	QUANTITY
BUFFER UPDATES	220.4K
PAGES WRITTEN	35169.00
BUFF.UPDATES/PAGES WRITTEN **B**	6.27
SYNCHRONOUS WRITES **D**	1003.00
ASYNCHRONOUS WRITES	5084.00
PAGES WRITTEN PER WRITE I/O **C**	5.78
HORIZ.DEF.WRITE THRESHOLD	2.00
VERTI.DEF.WRITE THRESHOLD	0.00
DM CRITICAL THRESHOLD **E**	0.00
WRITE ENGINE NOT AVAILABLE **F**	0.00
SYNC.HPOOL WRITE	0.00
ASYNC.HPOOL WRITE	5967.00
HPOOL WRITE FAILED	0.00
ASYN.DA.MOVER HPOOL WRITE-S	523.2K
ASYN.DA.MOVER HPOOL WRITE-F	0.00
PAGE-INS REQUIRED FOR WRITE	45.00

TOT4K SORT/MERGE	QUANTITY
MAX WORKFILES CONCURR. USED	0.00
MERGE PASSES REQUESTED	0.00
MERGE PASS DEGRADED-LOW BUF	0.00
WORKFILE REQ.REJCTD-LOW BUF	0.00
WORKFILE REQ-ALL MERGE PASS	0.00
WORKFILE NOT CREATED-NO BUF	0.00
WORKFILE PRF NOT SCHEDULED	0.00
WORKFILE PAGES TO DESTRUCT	4430.00
WORKFILE PAGES NOT WRITTEN	4430.00

Figure 102. DB2 PM Database Buffer Pool Statistics (Modified)

Tuning the EDM Pool

During the installation process, DSNTINST CLIST calculates the size of the EDM pool, based on parameters specified on the DSNTIPD and DSNTIPE panels.

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.

Refer to "Allied Thread Allocation" on page 5-116 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 *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 3 of *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 more information on the internal process that could increase the size of your plans, see "Considerations for Rebinding Certain Plans and Packages" on page 5-45. For information about how to estimate the size of the EDM pool, see *Installation Guide*.

Using Packages to Aid EDM Pool Storage Management

By using multiple packages you can increase the effectiveness of EDM pool storage management by decreasing the number of large objects in the pool.

Releasing thread storage

If your EDM pool storage grows continually, consider having DB2 periodically free unused thread storage. To do this, specify YES for the CONTSTOR subsystem parameter and then reassemble DSNTIJUZ. This option can affect performance and is best used when your system has many long-running threads and your EDM storage is constrained.

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 reprepared.
- Fewer threads used concurrently, due to a lack of storage

An EDM pool that is too large might use more virtual storage than necessary.

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 Section 2 of *Utility Guide and Reference*.

The DB2 statistics record provides information on the EDM pool. Figure 103 on page 5-68 shows how DB2 PM presents this information in the statistics report.

```
EDM POOL                     QUANTITY
--------------------------  --------
PAGES IN EDM POOL          A 16218.00
% PAGES IN USE                   6.07
FREE PAGES IN FREE CHAIN   B 15233.96
PAGES USED FOR CT               36.16
PAGES USED FOR DBD             136.36
PAGES USED FOR SKCT            755.71
PAGES USED FOR PT                4.41
PAGES USED FOR SKPT             51.40

FAILS DUE TO POOL FULL           0.00

REQUESTS FOR CT SECTIONS       135.1K
CT NOT IN EDM POOL             984.00
CT REQUESTS/CT NOT IN EDM  C   137.31
REQUESTS FOR PT SECTIONS     28302.00
PT NOT IN EDM POOL             134.00
PT REQUESTS/PT NOT IN EDM  D   211.21
REQUESTS FOR DBD SECTIONS    45799.00
DBD NOT IN EDM POOL             38.00
DBD REQUESTS/DBD NOT IN EDM E 1205.24

PREP_STMT_HIT_RATIO      F       0.67
PREP_STMT_CACHE_INSERTS          0.30
PREP_STMT_CACHE_REQUESTS         0.90
PREP_STMT_CACHE_PAGES_USED      47.11
```

Figure 103. 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:

CT REQUESTS/CT NOT IN EDM **C**
PT REQUESTS/PT NOT IN EDM **D**
DBD REQUESTS/DBD NOT IN EDM **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 IN FREE CHAIN **B** in Figure 103. 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.

QISEDSG records the number of requests to search the cache. QISEDSI 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. Use the following calculation to determine how often the dynamic statement was used from the cache:

```
(QISEDSG – QISEDSI) / QISEDSG = hit ratio
```

| The hit ratio is also shown in **F** in Figure 103.

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.

Increasing RID Pool Size

The RID pool is used for all record identifier (RID) processing. It is used for sorting RIDs during the following operations:

List prefetch, including single index list prefetch,
Access via multiple indexes
Hybrid joins

RID pool storage is also used when DB2 enforces unique keys while updating multiple rows.

SQL statements that use those methods of access can benefit from using the RID pool. RID pool processing can help reduce I/O resource consumption and elapsed time. However, if there is not enough RID pool storage, it is possible that the statement might revert to a table space scan.

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 1000MB. The RID pool is created at system initialization, but no space is allocated until RID storage is needed. It is then allocated above the 16MB line in 16KB 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 120KB 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.

When the DSNTINST CLIST calculates the value for RID POOL SIZE on panel DSNTIPC, the default is calculated as 50% of the sum of virtual buffer pools BP0, BP1, BP2, and BP32K.

You can modify the maximum RID pool size that you specified on installation panel DSNTIPC by using the installation panels in UPDATE mode, as follows:

- To favor the selection and efficient completion of list prefetch, multiple index access, or hybrid join, you can increase the maximum RID pool size.
- To disable list prefetch, multiple index access, and hybrid join, specify a RID pool size of 0.

  If you do this, 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.

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.

When the DSNTINST CLIST calculates the value for SORT POOL SIZE on panel DSNTIPC, the default is calculated as 10% of the sum of virtual BP0, BP1, BP2, and BP32K. The default is limited as follows:

```
MINIMUM = 240KB
MAXIMUM = 64000KB
```

You can change this value by using the installation panels in UPDATE mode. A rough formula for determining the maximum sort pool 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).

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.

Factors That Influence Sort Processing

You can influence the following factors that affect the performance of DB2 sort processing:

- Design your configuration to ensure minimal 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 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 5-41.

- The size of the sort pool affects the performance of the sort. The larger the work area, the more efficient the sort.
- When the sort occurs, the sort row size depends on the data fields that need to be sorted. Therefore, your 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 that can fit.
- VARCHARs are padded to their maximum length. Therefore, if VARCHAR columns are not required, your application should not select them. This will reduce the sort row size.

Other factors that influence sort performance include the following:

- The better sorted the data is, the more efficient the sort will be.
- 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 one, two, four, or eight pages. However, if the buffer pool is constrained, then prefetch could be disabled because not enough pages are available.
- 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 subselect, 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 5-5. 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 7 of *Data Sharing: Planning and Administration*.

This chapter covers the following topics:

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 simplifies the task of controlling resources for the user.

Each of the objectives presented in Table 62 on page 5-74 is matched with a control facility that you can use to achieve the objective. Each objective is then discussed separately in the sections that follow the table.

Table 62. Controlling the Use of Resources

Objective	How to Accomplish It
Prioritize resources	MVS dispatching priority, MVS workload management
Limit resources for each job	Time limit on job or step (through MVS or JCL)
Limit resources for TSO sessions	Time limit for TSO logon
Limit resources for IMS and CICS	IMS and CICS controls
Limit resources for a stored procedure	ASUTIME column of SYSIBM.SYSPROCEDURES catalog column.
Limit dynamic statement execution time	QMF governor and DB2 resource limit facility
Reduce locking contention	DB2 locking parameters, DISPLAY DB LOCKS, lock trace data, database design
Evaluate long-term resource usage	Accounting trace data, DB2 PM reports
Predict resource consumption	DB2 EXPLAIN statement, Visual Explain, and DB2 Estimator
Control use of parallelism	DB2 resource limit facility, SET CURRENT DEGREE statement (See “Disabling Query Parallelism” on page 5-313.)

Prioritize Resources

The **MVS dispatching priority** controls the execution of DB2 work. DB2 uses the processor and I/O dispatching priorities assigned to the requester's dispatchable unit (task or SRB) by MVS for all synchronous work done on behalf of the request. See *MVS/ESA Initialization and Tuning Guide* for information about assigning MVS processor and I/O dispatching priorities. Asynchronous I/O is governed by the MVS I/O dispatching priority of the address space requesting the I/O.

In CICS environments, DB2 work is performed in subtasks; therefore, the task-level processor dispatching priority also influences the ability of the DB2 subtasks to use the processor resource. 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 the MVS processor and I/O dispatching priorities assigned to the requester.

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 *MVS/ESA 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:

- *TSO/E Programming Guide*
- *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/ESA Operator's 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 on-line transactions. To establish limits for stored procedures, you can:

- Set a processor limit for each stored procedure, by updating the ASUTIME column of SYSIBM.SYSPROCEDURES catalog table. This 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 field MAX ABEND COUNT on installation panel DSNTIPX. This 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 5-329.

Limit Execution Time for Dynamic Statements

You can use either the QMF governor or the DB2 governor to set a limit on the available resources that a single SELECT, INSERT, UPDATE, or DELETE can use. The QMF governor handles only statements that are executed through QMF. The DB2 governor handles all dynamic SELECT, INSERT, UPDATE and DELETE
| statements. For cursor processing, the governor includes the OPEN and FETCH
| processing as part of the SELECT.

For more information on the DB2 governor, see “Resource Limit Facility (Governor)” on page 5-76. For more information about the QMF governor, see *Query Management Facility: Managing QMF for MVS*.

Reduce Locking Contention

DB2 uses *locking* to ensure the presentation of consistent data based on user requirements and to avoid losing data updates. You should provide a balance between concurrency, isolation, and resource usage. To accomplish this, you can separate data, choose lock parameters, and monitor for contention. For more information on locking contention, see “Chapter 5-7. Improving Concurrency” on page 5-137.

Evaluate Long-Term Resource Usage

You can use DB2 accounting trace data to control the amount of resources a group uses over a period of time. For example, you might want to ensure that a group does not exceed its budget or forecast. This type of control depends upon resource accounting. For more information about DB2 PM accounting records and DB2 PM reports, see “DB2 Performance Monitor (DB2 PM)” on page X-184.

For a more general discussion of accounting information, see *MVS/ESA Diagnosis: Procedures*.

Predict Resource Consumption

The best way to predict resource consumption for simple transactions and repetitive jobs is to measure it for a given period, and then use that measure as a basis for future forecasts. This method, however, does not work for dynamic queries.

You can use the EXPLAIN statement to understand the access path taken by an SQL statement. By reviewing the access path and the statistics for the index and data, you can estimate the expected resource usage for a given statement.

| Try also DB2 Estimator to do your capacity planning.

For more information on resource consumption, refer to:

“Initial Planning” on page 5-10
“Obtaining Information from EXPLAIN” on page 5-262
“Querying the Catalog for Statistics” on page 5-252.

Resource Limit Facility (Governor)

DB2's resource limit facility (governor) allows you to specify the maximum amount of processor time that any dynamic, manipulative SQL statement (SELECT, INSERT, UPDATE, and DELETE) from either the local DB2 subsystem or a remote DBMS can consume in DB2. See Chapter 7 of *Data Sharing: Planning and Administration* for information about special considerations for using the resource limit facility in a data sharing group.

The resource limit facility governs the execution of packages as well as plans. The resource limit facility also allows you to restrict bind and rebind operations. For example, you can disallow all bind activity during prime shift when production data is accessed. In addition, the DB2 governor allows you to restrict the use of
| parallelism. You can disallow CP parallelism, I/O parallelism, Sysplex query
| parallelism, or any combination of the three.

In some ways, the DB2 governor is very similar to the QMF governor; however, the two products actually complement each other. Before mixing the two governors,

determine your objectives for governing. Compare the differences in the implementation and operation of both, and decide when to use each facility to meet your governing requirements. The DB2 governor can limit the time spent processing interactive SQL statements, but it cannot interact with users. The QMF governor, on the other hand, governs all processor time consumed by the user address space, as opposed to only “in DB2” time.

The SELECT, INSERT, UPDATE, or DELETE statement can originate from your own location or from a remote DBMS, but the limit that you specify applies to your local processor. The resource limit facility does not control static SQL statements whether or not they are executed locally or remotely.

Queries that exceed the limit set by the governor are terminated with a -905 SQLCODE and a corresponding '57014' SQLSTATE. Termination prevents very long queries from exhausting your resources. You can establish a single limit for all users, different limits for individual users, or both. No limits apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority. Remote site resource limits apply when accessing remote system data through the distributed data facility.

One or more resource limit specification tables (RLSTs) define the limits. One resource limit specification table (the one identified on the -START RLIMIT command) is used each time the governor is invoked. You may, for example, want to define one RLST for prime shift and one for evening shift, as in Figure 104; 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 Section 2 of *Installation Guide* .

Prime Shift

SYSIBM.DSNRLST01			
AUTHID	PLANNAME	ASUTIME	LUNAME
BADUSER		0	LUDBD1
ROBYN		100000	LUDBD1
	QMF230	300000	LUDBD1
		50000	LUDBD1

Night Shift

SYSIBM.DSNRLST02			
AUTHID	PLANNAME	ASUTIME	LUNAME
BADUSER		0	LUDBD1
ROBYN		NULL	LUDBD1
	QMF230	NULL	LUDBD1
		300000	LUDBD1

Figure 104. Examples of RLST for Day and Night Shifts. During the night shift, AUTHID ROBYN and all QMF users from LUDBD1 run without limit.

Where RLSTs Reside

Resource limit specification tables can reside in any database, but because a database has some special attributes while the resource limit facility is active, we recommend that they reside in a database of their own.

When you install DB2, install job DSNTIJSG creates a table space for the resource limit specification tables by generating the following statements, which you can tailor:

```
CREATE DATABASE DSNRLST;
CREATE TABLESPACE DSNRLSxx IN DSNRLST CLOSE NO;
```

For more information about job DSNTIJSG, see Section 2 of *Installation Guide*.

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

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).

Creating an RLST

To define a resource limit specification table, use the following statements:

```
CREATE TABLE authid.DSNRLSTxx
       (AUTHID   CHAR(8)      NOT NULL WITH DEFAULT,
        PLANNAME CHAR(8)      NOT NULL WITH DEFAULT,
        ASUTIME  INTEGER,
        LUNAME   CHAR(8)      NOT NULL WITH DEFAULT,
        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)
       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.

When you create a resource specification table, it must include the AUTHID, PLANNAME, and ASUTIME columns, defined as they are in the above example. You can use the optional LUNAME column to specify limits for requests that originate at remote locations. The last four columns—RLFFUNC, RLFBIND, RLFCOLLN, and RLFPKG—are optional but mutually inclusive; any table that contains one of these columns must contain all of them. In addition, any resource limit specification table that contains RLFFUNC, RLFBIND, RLFCOLLN, and RLFPKG must also contain the LUNAME column. You can add any other columns you wish; they are ignored by the governor.

Table 63. Required and optional columns of a resource specification table

Column name	Required or Optional	Other columns required
AUTHID	required	none
PLANNAME	required	none
ASUTIME	required	none
LUNAME	optional	none
RLFFUNC	optional	LUNAME, RLFBIND, RLFCOLLN, RLFPKG
RLFBIND	optional	LUNAME, RLFFUNC, RLFCOLLN, RLFPKG
RLFCOLLN	optional	LUNAME, RLFFUNC, RLFBIND, RLFPKG
RLFPKG	optional	LUNAME, RLFFUNC, RLFBIND, RLFCOLLN

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.

You must also define a unique index on each RLST, using either of the following statements, depending upon whether or not you created the optional columns in your resource limit specification table. In either case, you must determine storage and buffer pool attributes for your location.

Without optional columns:

```
CREATE UNIQUE INDEX authid.DSNARLxx
       ON authid.DSNRLSTxx
         (AUTHID, PLANNAME)
       CLUSTER;
```

Only one entry with the same AUTHID.PLANNAME is allowed in the table.

With optional columns:

```
CREATE UNIQUE INDEX authid.DSNARLxx
       ON authid.DSNRLSTxx
         (RLFFUNC, AUTHID, PLANNAME, RLFCOLLN, RLFPKG, LUNAME)
       CLUSTER CLOSE NO;
```

Only one entry is allowed in the table with the same RLFFUNC.AUTHID.PLANNAME.RLFCOLLN.RLFPKG.LUNAME. The *xx* in the index name (DSNARL*xx*) must match the *xx* in the table name (DSNRLST*xx*). Further, the index for the RLST *must* be ascending, which the previous sample SQL statements provide.

The creator of the tables and indexes must have sufficient authority to define objects in the DSNRLST database, and to specify the authorization identifier of *authid* in the definition.

Populating the Resource Limit Specification Table: 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.

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.

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.

What the RLST Contains

Each row in a resource limit specification table contains the following:

AUTHID The resource specification limits apply to this primary authorization ID. To have the limit specifications in this row apply to all authids for the location specified in LUNAME, this column must contain blank. If LUNAME is blank, or the column is not included in the table, then the location is the local location.

PLANNAME The resource specification limits apply to this plan. A blank value in this column means that the limit specifications in this row apply to all plans for the location specified in LUNAME. If LUNAME is blank, or the column is not included in the table, then the location is the local location. If the RLFFUNC column is present and contains a `'1'` or `'2'`, then this column must be blank; if it is not blank the row is ignored.

ASUTIME The number of system resource manager processor service units permitted to any single dynamic SELECT, INSERT, UPDATE, or DELETE statement. A null value in this column means there is no limit. A zero or a negative value means that no dynamic SELECT, INSERT, UPDATE, or DELETE statements are permitted.

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 have to be DB2 subsystems.

RLFFUNC Specifies how the row is used:

- `'1'` means that the row governs bind operations.
- `'2'` means that the row governs dynamic DML by package or collection name.
- `'3'` means that the row disables query I/O parallelism.
- `'4'` means that the row disables query CP parallelism.
- `'5'` means that the row disables Sysplex query parallelism.
- Blank means that the row governs dynamic DML statements by plan name.
- All other values are ignored.

RLFBIND Shows whether or not 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 The resource limit specifications apply to this package collection. A blank value in this column means that the limit specifications apply to all package collections from the location specified in LUNAME. If LUNAME is blank, or the column is not included in the table, then the location is the local location. If RLFFUNC=blank or RLFFUNC=`'1'`, then RLFCOLLN must be blank. If RLFFUNC=`'2'`, then the resource limits apply to the package collection named in this column.

RLFPKG The resource limit specifications apply to this package. A blank value in this column means that the limit specifications apply to all packages from the location specified in LUNAME. If LUNAME is blank, or the column is not included in the table, then the location is the local location. If RLFFUNC=blank or RLFFUNC='1', then RLFPKG must be blank. If RLFFUNC='2', then the resource limits apply to the package named in this column.

Processor Service Units (ASUTIME): The value for ASUTIME, which is specified
in service units rather than just processor time, is independent of processor
changes. The governor samples the processing time in service units. 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.

| To calculate ASUTIME, use the following formula:

```
| ASUTIME = 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 *MVS/ESA 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 = 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 for your current processor, there is no need to modify it when you change processors.

Understanding RLST Search Order and Column Combinations

Table 64 on page 5-82, Table 65 on page 5-83, and Table 66 on page 5-83 show the search order, or precedence, of entries in the RLST for each method of governing. If more than one entry exists with the same AUTHID and the same PLANNAME, then the value of LUNAME determines the precedence order. The value of LUNAME also determines the precedence order when more than one entry exists with the same AUTHID and a blank PLANNAME.

If no row is found in the RLST, the default limit set at installation time for either local or remote dynamic SELECT, INSERT, UPDATE, or DELETE statements is used, according to the type of statement. You can set the default limits on panel

DSNTIPR (for statements originating at the remote location) or on panel DSNTIPO (for statements originating at the local location).

Table 64. Search Order of Entries in RLST When Governing Bind Operations

RLFFUNC	AUTHID	LUNAME	Governs bind operations for
1	Name	Name	A specific AUTHID at a specific location
1	Name	Blank	A specific AUTHID at the local location
1	Name	PUBLIC	A specific AUTHID at all locations
1	Blank	Name	All AUTHIDs at a specific location
1	Blank	Blank	All AUTHIDs at the local location
1	Blank	PUBLIC	All AUTHIDs at all locations.

Note: When governing bind authority, columns PLANNAME, RLFCOLLN, and RLFPKG *must* be blank.

Governing by Plan Name or Package Name: Governing by plan name and package name are mutually exclusive.

- In governing by plan name, the RLF governs the DBRMs in the MEMBER list specified on the BIND PLAN command. To govern by plan name, your RLST should contain blanks for columns RLFFUNC, RLFCOLLN, and RLFPKG.
- In governing by package name, the RLF governs the packages used during the execution of the SQL application program. To govern by package name, your RLST should contain a row with PLANNAME=BLANK and RLFFUNC=2.

RLST Combinations for SPUFI: Because SPUFI is bound as a package, SPUFI users must have an RLST row with the values RLFFUNC=2 and PLANNAME=BLANK. The values for RLFCOLLN and RLFPKG can be blank.

RLST Combinations for Distributed Processing: For distributed processing, keep in mind the following guidelines:

- When dynamic SELECT, INSERT, UPDATE, or DELETE statements are issued from a remote location to access data at your local location using DRDA protocols, they are governed by a row in the RLST with RLFFUNC=2 and LUNAME=originating location first, LUNAME=PUBLIC second. The value in the PLANNAME column must be blank, but you can specify values for RLFPKG and RLFCOLLN.
- When dynamic SELECT, INSERT, DELETE and UPDATE statements are issued from a remote location to access data at your local location using DB2 private protocol access, they are governed by a row in the RLST with RLFFUNC=Blank and LUNAME=originating location first, LUNAME= PUBLIC second. For DB2 Version 2 Release 3, only PLANNAME=blank is valid. For DB2 Version 3 and above, PLANNAME can be blank or the name of the plan created at the requester's location. RLFPKG and RLFCOLLN must be blank.
- If no row is present in the RLST to govern access from a remote location, the limit is the default set on panel DSNTIPR at installation time.

Table 65. Search Order of Entries in RLST When Governing by Plan Name

RLFFUNC	AUTHID	PLANNAME	LUNAME	Limit Applies to
Blank	Name	Name	Name	A specific AUTHID for the named PLANNAME at a specific location
Blank	Name	Name	Blank	A specific AUTHID for the named PLANNAME at the local location
Blank	Name	Name	PUBLIC	A specific AUTHID for the named PLANNAME at all locations
Blank	Name	Blank	Name	A specific AUTHID for all PLANNAMEs at a specific location
Blank	Name	Blank	Blank	A specific AUTHID for all PLANNAMEs at the local location
Blank	Name	Blank	PUBLIC	A specific AUTHID for all PLANNAMEs at all locations
Blank	Blank	Name	Name	All AUTHIDs for the named PLANNAME at a specific location
Blank	Blank	Name	Blank	All AUTHIDs for the named PLANNAME at the local location
Blank	Blank	Name	PUBLIC	All AUTHIDs for the named PLANNAME at all locations
Blank	Blank	Blank	Name	All AUTHIDs for all PLANNAMEs at a specific location
Blank	Blank	Blank	Blank	All AUTHIDs for all PLANNAMEs at the local location
Blank	Blank	Blank	PUBLIC	All AUTHIDs for all PLANNAMEs at all locations

Note: When governing by plan name, columns RLFFUNC, RLFCOLLN, and RLFPKG *must* be blank.

Table 66 (Page 1 of 2). Search Order of Entries in RLST When Governing by Collection and Package Name

RLFFUNC	AUTHID	LUNAME	RLFCOLLN	RLFPKG	Limit Applies to
2	Name	Name	Name	Name	A specific AUTHID for a specific collection and package at a specific location
2	Name	Blank	Name	Name	A specific AUTHID for a specific collection and package at the local location
2	Name	PUBLIC	Name	Name	A specific AUTHID for a specific collection and package at all locations
2	Name	Name	Name	Blank	A specific AUTHID for a specific collection and all packages at a specific location
2	Name	Blank	Name	Blank	A specific AUTHID for a specific collection and all packages at the local location
2	Name	PUBLIC	Name	Blank	A specific AUTHID for a specific collection and all packages at all locations
2	Name	Name	Blank	Name	A specific AUTHID for all collections and a specific package at a specific location
2	Name	Blank	Blank	Name	A specific AUTHID for all collections and a specific package at the local location

Table 66 (Page 2 of 2). Search Order of Entries in RLST When Governing by Collection and Package Name

RLFFUNC	AUTHID	LUNAME	RLFCOLLN	RLFPKG	Limit Applies to
2	Name	PUBLIC	Blank	Name	A specific AUTHID for all collections and a specific package at all locations
2	Name	Name	Blank	Blank	A specific AUTHID for all collections and packages at a specific location
2	Name	Blank	Blank	Blank	A specific AUTHID for all collections and packages at the local location
2	Name	PUBLIC	Blank	Blank	A specific AUTHID for all collections and packages at all locations
2	Blank	Name	Name	Name	All AUTHIDs for a specific collection and package at a specific location
2	Blank	Blank	Name	Name	All AUTHIDs for a specific collection and package at the local location
2	Blank	PUBLIC	Name	Name	All AUTHIDs for a specific collection and package at all locations
2	Blank	Name	Name	Blank	All AUTHIDs for a specific collection and all packages at a specific location
2	Blank	Blank	Name	Blank	All AUTHIDs for a specific collection and all packages at the local location
2	Blank	PUBLIC	Name	Blank	All AUTHIDs for a specific collection and all packages at all locations
2	Blank	Name	Blank	Name	All AUTHIDs for all collections and a specific package at a specific location
2	Blank	Blank	Blank	Name	All AUTHIDs for all collections and a specific package at the local location
2	Blank	PUBLIC	Blank	Name	All AUTHIDs for all collections and a specific package at all locations
2	Blank	Name	Blank	Blank	All AUTHIDs for all collections and packages at a specific location
2	Blank	Blank	Blank	Blank	All AUTHIDs for all collections and packages at the local location
2	Blank	PUBLIC	Blank	Blank	All AUTHIDs for all collections and packages at all locations

Note: When governing by collection and package name, column PLANNAME *must* be blank.

Limit Values: If you do not have a limit for a combination of AUTHID and PLANNAME, or for a combination of AUTHID, RLFCOLLN, and RLFPKG, set ASUTIME to null. If you make the limit value less than or equal to zero, you disable operations for the specified combinations, because no time is permitted for dynamic SELECT, INSERT, UPDATE, or DELETE statements. A positive value limits the number of service units that a dynamic DML statement can use.

Locking Considerations for RLSTs: While the governor is active, DB2 issues an implicit SELECT statement from the active resource limit specification table when a job issues its first dynamic SELECT, INSERT, UPDATE, or DELETE statement. The implicit SELECT acquires standard DB2 locks on the RLST and its table space. This process could have an effect on any INSERT, UPDATE, or DELETE

statements issued for the RLST, or for any other table in the same table space as the active RLST.

For more information about DB2 locks, see “Chapter 5-7. Improving Concurrency” on page 5-137.

RLST Combinations for Parallel Processing: When 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 two rows in your RLST. One row with RLFFUNC='3' and one row with RLFFUNC='4' for query I/O parallelism and query CP parallelism, respectively. If you have data sharing and have enabled Sysplex query parallelism, then an additional row containing RLFFUNC='5' disables that. See Chapter 7 of *Data Sharing: Planning and Administration* for more information about how governing works with Sysplex query parallelism.
- If no entry can be found in your RLST which applies to parallelism, or if your RLST cannot be read, then the resource limit facility does not disable query parallelism.
- Parallelism modes cannot be disabled for static statements through the resource limit facility.

In the example in Table 67 the resource limit facility searches the active RLST to determine whether or not the user has been disabled for any type of query parallelism.

Table 67 (Page 1 of 2). Example RLST to Govern Query Parallelism. The authorization ID BADGUY is not allowed to do any type of query parallelism.

RLFFUNC	AUTHID	LUNAME	PLANNAME	RLFCOLLN	RLFPKG	Effect
3	IOHOG	PUBLIC	blank	blank	blank	Disables query I/O parallelism for AUTHID IOHOG for all plans and all packages in all collections at all locations.
4	CPUHOG	PUBLIC	blank	blank	blank	Disables query CP parallelism for AUTHID CPUHOG for all plans and all packages in all collections at all locations.
5	CPUHOG	PUBLIC	blank	blank	blank	Disables Sysplex query parallelism for AUTHID CPUHOG for all plans and all packages in all collections at all locations.

Table 67 (Page 2 of 2). Example RLST to Govern Query Parallelism. The authorization ID BADGUY is not allowed to do any type of query parallelism.

RLFFUNC	AUTHID	LUNAME	PLANNAME	RLFCOLLN	RLFPKG	Effect
3	blank	blank	blank	GOODCPU	blank	Disables query I/O parallelism for all AUTHIDs for all packages in the GOODCPU collection at the local location.
4	blank	blank	GOODIO	blank	blank	Disables query CP parallelism for all AUTHIDs for the GOODIO plan at the local location.
3	BADGUY	PUBLIC	blank	blank	blank	Disables query I/O parallelism for AUTHID BADGUY for all plans and all packages in all collections at all locations.
4	BADGUY	PUBLIC	blank	blank	blank	Disables query CP parallelism for AUTHID BADGUY for all plans and all packages in all collections at all locations.
5	BADGUY	PUBLIC	blank	blank	blank	Disables Sysplex query parallelism for AUTHID BADGUY for all plans and all packages in all collections at all locations.

Using RLSTs at Your Local Subsystem

If you are a system administrator, you must determine how your location intends to use the governor and create several local procedures. You should establish a procedure for creating and maintaining your RLSTs, and for establishing limits for any newly written applications. You also need to devise required procedures for console operators, such as switching RLSTs every day at a certain time.

When dynamic SELECT, INSERT, UPDATE, or DELETE statements are issued from your location to access data at a remote location using DB2 private protocol, they are governed by RLSTs at both locations.

In application programs that use dynamic SELECT, INSERT, UPDATE, or DELETE statements, each statement in the program is subject to the same time limit, which is the limit that exists when the job makes its first dynamic SQL request. If any statement exceeds the limit, the statement is not executed, and a unique SQL error code is returned. Code your application programs so that they continue a course of action if a statement exceeds the limit.

If the failed instruction has a cursor associated with it, the cursor's position is unchanged, and the application can close the cursor. Any other operations that have been made with that cursor are not executed, and the same SQL error code is returned.

If the failed SQL statement does not have a cursor associated with it, all changes the statement made are lost before the error code is returned to the application, but the unit of work previous to that statement is not lost. The application can then issue another SQL statement, or commit all work done so far.

For more information on the commands used to start and stop the governor, see Chapter 2 of *Command Reference*.

Managing the Opening and Closing of Data Sets

The number of open data sets affects the amount of available storage as well as restart and shutdown times. However, is it important for the performance of transactions that needed data sets are open and available for use. This section describes how this open and close activity is managed by DB2, and gives some recommendations about how you can influence this processing. The following topics are described:

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. So DSMAX controls not only the limit of open data sets, but also how many data sets 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 partitions into account. If you have many partitioned table spaces, then evaluate DSMAX more carefully. A nonpartitioning index on a table space defined as LARGE can have up to 128 data sets; if you are using a small PIECESIZE for nonpartitioned indexes, you might need to increase DSMAX to account for those extra 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 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. (This assumes one data set per each simple or segmented table space.)

   These catalog queries are included in DSNTESP in SDSNSAMP and can be used 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” on page 5-90.

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.
- For the best restart times after an abnormal termination, there can be advantages in limiting the number of open data sets. A smaller number of open data sets means there are fewer data sets to open when you restart DB2 after an abnormal termination.
- The number of open data sets on your subsystem that are in read/write state affects checkpoint costs and log volumes. Use the PCLOSEN and PCLOSET subsystem parameters to control how long data sets stay open in a read/write state. See “Switching to Read-Only for Infrequently Updated Page Sets” on page 5-91 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. For more information on using segmented table spaces, see “Creating a Segmented Table Space” on page 2-90.

Understanding the CLOSE YES and CLOSE NO Options

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. We use the term **page set** to refer to a table space or index.

The Process of Closing

DB2 dynamically manages page sets using 2 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 (not an index) 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 5-87, it is the number of open data sets that 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.

It is the value you specify for CLOSE that 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 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. You should consider having a "priming job" that accesses these crucial tables when you start DB2. This insures that these crucial page sets are always open when DB2 is running.

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 parameters PCLOSEN and PCLOSET. PCLOSEN is the number of consecutive DB2 checkpoints since a page set or partition was last updated; the default is 5. PCLOSET is the amount of elapsed time since a page set or partition was last updated; the default is 10 minutes. If either the PCLOSEN or PCLOSET 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 for and any updated pages are externalized to DASD. For indexes, there is no SYSLGRNX entry, but the updated pages are externalized to DASD.

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 PCLOSET and PCLOSEN: 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 PCLOSET, perhaps to 30 minutes.

To change PCLOSEN or PCLOSET, update the values in the DSNTIJUZ installation job, run DSNTIJUZ, and restart DB2.

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.

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 + 300 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 G. Using Tools to Monitor Performance” on page X-173 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 data base, 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 Section 2 of *Utility Guide and Reference*.

Monitoring I/O Activity of Data Sets

One way to monitor data set I/O activity is to use the DB2 PM report titled I/O Activity Report - Buffer Pool. 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 5-40 and 5-29.

For each database and page set, this report shows the:

- Total number of I/Os
- Average time required for reads
- Total number for each type of read request (such as dynamic prefetch, list prefetch, sequential prefetch and synchronous)
- Average of the times required for reads by type of read
- The percentage of requests that required I/O by type of read
- Number of pages read per synchronous I/O or per prefetch request
- Percentage of prefetch requests without I/O. This can occur when all of the requested pages are in the buffer pool.
- Number of synchronous and asynchronous write requests
- Average time required for writes
- Average number of pages written per request

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 Tables: Section 2 of *Installation Guide* contains information about how to estimate the DASD storage required for temporary tables. The storage is similar to that required for regular tables. When a temporary 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 table until the temporary 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 temporary tables, keep work files in a separate buffer pool. Use IFCID 0311 in performance trace class 8 to distinguish temporary table work 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 4-3. Managing the Log and the Bootstrap Data Set” on page 4-83. For the sake of performance, it is important to understand the log requests that an application program makes, and their effect on the number of I/Os.

Logging Options: Specify logging options at installation time on the Log Data Sets panel DSNTIPL. On this panel, the most important options for tuning performance are OUTPUT BUFFER and WRITE THRESHOLD.

- The OUTPUT BUFFER field specifies the size of the output buffer used for writing active log data sets. The maximum size of OUTBUFF is 4000 KB. The buffer must be large enough to prevent buffer shortages in order to decrease the number of forced I/O operations (forced because there are no more buffers) or wait conditions. We recommend that you use the maximum allowed value for this field.
- The WRITE THRESHOLD field indicates the number of contiguous 4KB output buffer pages that are allowed to fill before data is written to the active log data set. The default is 20. We recommend that you use the default for this field. Never choose a value that is greater than 20% of the number of buffers in the output buffer.
- Under normal circumstances, use the default values for the fields NUMBER OF LOGS, OUTPUT BUFFER, WRITE THRESHOLD and INPUT BUFFER.

DB2 uses two different types of requests when it logs data and events:

NOWAIT: NOWAIT requests are the most frequent. When data is updated, before- and after-image records are usually moved to the log buffer, and control is returned to the application. However, if no log buffer is available, the application must wait for one. When the number of log buffers used reaches the WRITE THRESHOLD value, the data is written; however, the application does not wait for the write.

Batch jobs might log a large amount of data before they commit. The amount of data logged between FORCE requests can be larger than the total buffer size. If so, we recommend that the write threshold (WRTHRSH) not exceed 20% of the total number of buffers in the pool. This size allows DB2 to continue using buffers while the filled buffers are being written to the active log data sets.

FORCE: FORCE requests occur at commit time, when an application has performed database updates. If the log data set is not busy, all log buffers are written to disk. The application must wait until the write is completed. If the log data set is busy, the requests are queued until it is freed.

CICS and IMS attachment facilities use two FORCE requests during a commit; TSO uses only one.

By the time a program reaches a commit point, the database change records might already have been written, if the log buffer write threshold has been reached or another process has made a FORCE request. Otherwise, the records are written at commit time, as shown in Figure 105 on page 5-95.

Figure 105 on page 5-95 shows that during two-phase commit, the two FORCE log requests cause two waits. The first FORCE request forces all the log records of changes to be written, if they have not already been written. When two logs are used, the write to the first log must be completed before the write to the second log begins.

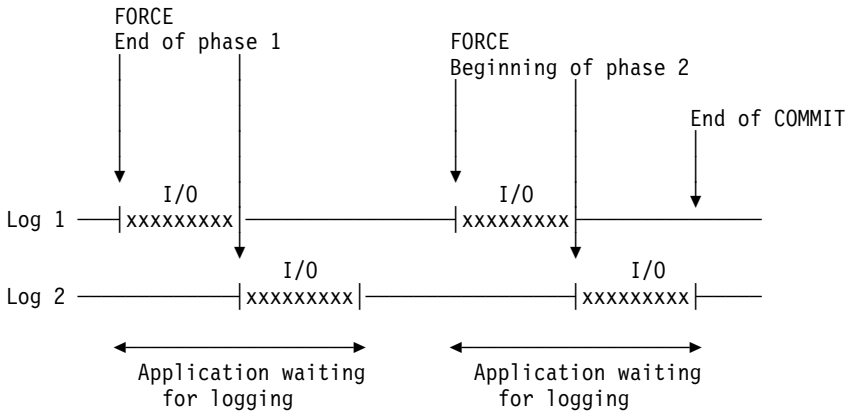


Figure 105. Dual Logging during Two-Phase Commit

You can design the DB2 log for better performance. In particular, try to:

- Minimize device contention on the log data sets by placing data sets correctly.
- Avoid waits that occur because no log buffer is available.
- Define enough active log data sets to prevent DB2 from waiting while a log is archived.
- Make the active log large enough that backouts do not have to use the archive log.
- Whenever you allocate new active log data sets, preformat them using the DSNJLOGF utility described in Section 3 of *Utility Guide and Reference*. This avoids the overhead of preformatting the log, which normally occurs at unpredictable times.

Log Devices: Because the commit process is synchronized with the active log writes, the devices assigned to the active log data sets must be fast ones. In general, log data sets can make effective use of the DASD Fast Write feature of IBM's 3990 cache.

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. DB2 writes serially or in parallel to the log data sets. If a 4KB log control interval is written for the first time to DASD, the write I/Os to the log data sets are done in parallel. If the same 4KB log control interval is again written to DASD, 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. This improves system integrity; there is no I/O overlap in dual logging, except when the WRITE THRESHOLD value is reached, triggering full output log buffers to be written, see Figure 105. For more information on volume considerations, refer to Section 2 of *Installation Guide* .

I/O Contention on Log Data Sets:Avoid device contention on 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 the 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.

Determining the Size of Active Logs

The total capacity provided for the active log can affect DB2 performance significantly. Four 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 Section 2 of *Installation Guide* for more information on updating the active log parameters. The parameters are listed below:

- NUMBER OF LOGS on the Log Data Sets Panel (DSNTIPL) controls the number of active log data sets you create.
- ARCHIVE LOG FREQ on the Log Data Sets Panel (DSNTIPL) provides an estimate of how often active log data sets are copied to the archive log.
- UPDATE RATE on the Log Data Sets Panel (DSNTIPL) estimates the number of database changes (inserts, updates, and deletes) expected per hour.
- CHECKPOINT FREQ on the Operator Functions Panel (DSNTIPN) specifies the number of log records that DB2 writes between checkpoints.

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 and restart. This takes a considerable amount of time. When you are deciding how many active log data sets to use and how large each must be, consider that:

- Performance is negatively affected when DB2 must access data on an archive log data set. This can occur during rollback or restart. 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).

  During rollback, DB2 does not share access to an archive log data set with multiple units of work. In other words, one unit of work retrieves all records from an archive log data set before another unit of work is allowed to access that archive log data set. Database locks, virtual storage, and other resources are not freed until the rollback completes. DB2 is unavailable for new work until restart processing is complete.

  However, if the archive log data set resides on DASD, it can be shared by many units of work.
- 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. Provide enough active log space for 10 checkpoint intervals. For estimation purposes, assume that a single checkpoint writes 24KB (or 6 control intervals) of data to the log. A checkpoint interval is defined by the number you specify for checkpoint frequency (see Section 2 of

Installation Guide). Make sure that the number you specify multiplied by 10 is less than the number of changes per hour multiplied by the number of hours per archive.

- When selecting values for the parameters described above, avoid creating a few very large active log data sets. This happens if you specify a large number for ARCHIVE LOG FREQ on the Log Data Sets Panel (DSNTIPL), and a small number for NUMBER OF LOGS (also found on panel DSNTIPL). This often causes active log data set shortages.
- Avoid creating many very small data sets. This happens if you specify a small number for ARCHIVE LOG FREQ and a large number for NUMBER OF LOGS. This causes operational problems if you archive to tape.
- When archiving to tape, remember that the archive log contains the BSDS. You can create an archive log data set that spans up to 20 tape volumes.
- When archiving to DASD, set the primary space quantity and block size for the archive log data set so you can off-load the active log data set without forcing the use of secondary extents in the archive log data set. This avoids space abends when writing the archive log data set.
- We recommend that the number of records for the active log be divisible by the blocking factor of the archive log (DASD or tape).

  To determine the blocking factor of the archive log, divide the value specifed 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 for the active log data set is a multiple of the blocking factor.

  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 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.
- When you calculate the size of the 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 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.
- The active log data set should be large enough to hold the data written while the previous active logs are archived. When you estimate the time required for archiving, also 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.
- If you off-load 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.

For more information on determining and setting the size of your active log data sets, refer to *Installation Guide*.

Monitoring the Log

DB2 PM provides statistics about the DB2 log, as shown in Figure 106. In this report, note especially the following fields:

READS SATISFIED FROM ARCHIVE LOG **A**: For optimal performance, when data is backed out, it should still be available in the output buffer or in the active log. If the data has already been off-loaded to the archive log, the active log is probably too small.

UNAVAILABLE OUTPUT LOG BUFF **B**: This field shows how many times a write request to the active log waited because no buffer was available. These waits should not occur. If these waits do occur, the output buffer might be too small, or the size of the write threshold might be too close to the size of the output buffer.

```
LOG ACTIVITY                QUANTITY
--------------------------  --------
READS SATISFIED-OUTPUT BUFF    269.00
READS SATISFIED-OUTP.BUF(%)     98.18
READS SATISFIED-ACTIVE LOG       5.00
READS SATISFIED-ACTV.LOG(%)      1.82
READS SATISFIED-ARCHIVE LOG  A   0.00
READS SATISFIED-ARCH.LOG(%)      0.00
TAPE VOLUME CONTENTION WAIT      0.00

WRITE-NOWAIT                   298.9K
WRITE OUTPUT LOG BUFFERS     19200.00

BSDS ACCESS REQUESTS           377.00
UNAVAILABLE OUTPUT LOG BUFF  B   0.00

CONTR.INTERV.CREATED-ACTIVE   6980.00
ARCHIVE LOG READ ALLOCATION      0.00
ARCHIVE LOG WRITE ALLOCAT.       0.00
CONTR.INTERV.OFFLOADED-ARCH      0.00

READ DELAYED-UNAVAIL.RESOUR      0.00
READ DELAYED-ARCH.ALLOC.LIM       N/A
LOOK-AHEAD MOUNT ATTEMPTED       0.00
LOOK-AHEAD MOUNT SUCCESSFUL      0.00
```

Figure 106. Log Statistics in the DB2 PM Statistics Report

Guidelines for Controlling Logging

Certain processes cause a large amount of information to be logged, requiring a large amount of log space. To avoid wasting tape media, consider the following:

- 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.

As a guideline, specify LOAD LOG(YES) when adding a small percentage of records. This creates additional logging, but eliminates the need for a full image copy. When loading many records or issuing a REORG on large tables, specify LOG(NO) and take a full image copy immediately after the LOAD or REORG (or use inline copy) .

- 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 subselect from another table
  - DELETE/UPDATE

  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, the volume of log data written for mass DELETE statements is much less than for nonsegmented table spaces.

  When loading a large amount of data, use the LOAD utility rather than the SQL INSERT statement whenever possible. This allows you to control the logging performed for the utility. Because the utility takes intermediate commit points, the demand for log information at rollback and restart is minimized.

  Try to use segmented table spaces or create one table per table space. For nonsegmented table spaces, use the SQL DROP statement to drop the entire table space rather than delete all rows or drop the table itself. This operation does not log the data involved.
- Another way to reduce log volume for updated variable-length columns (including those that are compressed) is to place the updated columns near the end of the row. DB2 logs from the first changed byte to the end of the record. However, if you have columns defined as variable-length and they do not get updated, keep in mind that DB2 accesses fixed length columns faster than variable length columns. Generally, you should have fixed length columns precede variable length columns because of the access time. Base your optimization decisions on your specific environment.

Improving DASD Utilization: Space and Device Utilization

To use DASD **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” on page 5-100.
- Manage them with the Hierarchical Storage Management functional component (DFSMShsm) of DFSMS/MVS, as described in “Using DFSMShsm to Manage Data Sets” on page 5-101.
- Compress your data, as described in “Compressing Your Data” on page 5-102.

To manage the **use** of DASD, you can use RMF to monitor how your devices are used. Watch for usage rates that are higher than 30% to 35%, and for 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 DASD 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, you might have more extends to satisfy those activities that need a large space.

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. The added volumes are only those that have not been used by DB2 for the data set. 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.

Extending Nonpartitioned Spaces: For a nonpartitioned table space or 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. However, to do the extension, the value of (PRIQTY + 118 × SECQTY) must be at least 2GB for nonpartitioned table spaces. For nonpartitioning indexes, that value must reach the user-specified PIECESIZE or the default of 2GB. If DB2 reaches the maximum number of extents (119) before reaching the PIECESIZE or 2GB limit, the extension fails.

For nonpartitioning indexes on partitioned table spaces defined as LARGE, the value of PRIQTY + 118 × SECQTY must be at least the user-specified value of PIECESIZE, or the default value of 4GB. If DB2 reaches the maximum number of extents before reaching the PIECESIZE or 4GB limit, the extension fails.

Monitoring Data Set Extensions: Use IFCID 0258 in statistics class 3 to monitor extension activity. The record is written whenever a data set is extended, and it includes such information as the primary and secondary quantity values, the maximum number of extents this data set has reached, and the number of extents both before and after the current extension.

Extending User-Managed Data Sets

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

Using DFSMShsm to Manage Data Sets

The Hierarchical Storage Management functional component (DFSMShsm) of DFSMS/MVS 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 DASD space is managed efficiently.

The storage manager reviews data storage daily. DFSMShsm can delete data sets or move them to another device. The space manager specifies the time when it will perform these tasks, as well as how long data sets are kept before they are deleted or moved. You can initiate many DFSMShsm operations directly, rather than waiting for the assigned processing window.

The following are some features of DFSMShsm and guidelines for using DFSMShsm with DB2:

- 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.

- DB2 table spaces and index spaces can also be automatically migrated and recalled.

- If you accepted the default value YES for the RECALL DATABASE parameter on the Operator Functions panel (DSNTIPO), DB2 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 temporary databases, can reside on system managed storage (SMS) but should be recalled via DFSMShsm before starting DB2. An alternative is to avoid migrating by assigning a management class to these data sets that prevents migration.

- 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.

For more information about DFSMShsm, see *DFSMS/MVS: DFSMShsm Managing Your Own Data*.

Compressing Your Data

Using the COMPRESS clause of the CREATE TABLESPACE and ALTER TABLESPACE SQL statements allows you to compress data in a table space or in a partition of a partitioned table space. In many cases, using the COMPRESS clause can significantly reduce the amount of DASD space needed to store data, but the compression ratio you achieve depends on the characteristics of your data. In addition, with compressed data, the amount of I/O required to scan a table space can be reduced.

For information on using compressed data, see “Compressing Data in a Table Space or Partition” on page 2-63.

Performance Considerations

To estimate how well a given table space will compress, run the standalone utility DSN1COMP. Consider the following when making your decision:

- 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. Indexes are not compressed.
- The processor cost to decode a row using the COMPRESS clause is significantly less than the cost to encode that same row. This rule applies regardless of whether the compression uses the synchronous data compression hardware or the software simulation built into DB2.
- When rows are accessed sequentially, fewer I/Os might be required to access data stored in a compressed table space. However, there is a trade-off 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.
- Depending on the SQL work load, as the degree of data compression increases, it can lead to:
  - Higher buffer pool hit ratios
  - Fewer I/Os
  - Fewer getpage operations
- The data access path DB2 uses affects the processor cost for data compression. In general, the relative overhead of compression is higher for table space scans and less costly for index access.

- 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 of the high frequency of certain byte strings.
- In determining whether using the COMPRESS clause is advantageous, examine the uncompressed length of a row. 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 32KB page can enhance the benefits of compression, especially if the data is accessed primarily in a sequential mode.

Tuning Recommendations

In some cases using compressed data results in an increase in the number of getpages, lock requests, and synchronous read I/Os. There are two possible reasons for this:

- 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, there is a great probability that an updated row 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.

- With type 1 indexes, data compression can cause more lock requests with the same number of getpages that would occur with no data compression. With datacompressed, more rows are stored in a page. More rows means that DB2 must lock more index pages per data page. This action has a negligible performance effect.

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, refer to Section 2 of *Installation Guide*. If you use the distributed data functions of DB2, you will probably find that you need more virtual storage. You can expect your storage needs to increase in the extended common storage area (ECSA) above the 16MB line by:

- 1KB for each conversation, plus
- 2KB for each thread that uses distributed processing, plus
- 1KB for each DB2 site in your network, plus
- 40KB for code that relates to distributed processing

Your virtual storage needs will probably not increase in the CSA below the 16MB line. Any other increase in the amount of virtual storage needed occurs within the

extended private area of the DB2 database address space and the extended private area of the distributed data address space.

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 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, use type 2 indexes, or use cursor stability with CURRENTDATA NO. For more information on specifying LOCKSIZE TABLESPACE, see “Monitoring DB2 Locking” on page 5-191.

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 4KB page. For information about this, see “Make Buffer Pools Large Enough for the Work Load” on page 5-40.

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. For suggestions on reducing the need for sort, see “Determining Sort Activity” on page 5-294.

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 Section 2 of *Installation Guide*.

Ensure EDM Pool Space Is Being Used Efficiently: Monitor your use of EDM pool storage using DB2 statistics; see “Tuning the EDM Pool” on page 5-66.
Reduce the value for EDMPOOL storage if possible. See “Releasing thread
storage” on page 5-66 for another option on controlling EDM pool storage.

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, expanded storage, 3990 cache, DASD, optical storage, and tape or cartridge.

Central Storage

Central storage refers to the processor storage where program instructions reside while they are executing. Data in DB2's virtual buffer pools resides in central storage only or in central and expanded storage, and thus provides the fastest data access because no I/O operations are required. The maximum amount of central storage that one DB2 subsystem can use is about 2GB.

Expanded Storage

Expanded storage is optional high-speed processor storage. Data is moved in 4KB 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 running under MVS Version 4 Release 3, or later releases, and the Asynchronous Data Mover Facility of MVS is installed, DB2 can use up to 8GB of expanded storage by creating hiperpools. For more information on how DB2 uses hiperpools, see “Buffer Pools and Hiperpools” on page 5-49.

3990 Cache

DB2 can take advantage of cache in the 3990 Model 3 and the 3990 Model 6 Storage Controls. To understand how DB2 can use 3990 cache, you need to understand how cache storage fits into the storage hierarchy. DB2's primary “cache” is the central storage and expanded storage in the processor. Storage pools that reside in processor storage include:

- Hiperpools
- Buffer pools
- Buffers for log data
- EDM pool for database descriptors, plans, and packages
- RID pool
- 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 DASD devices are used more effectively.

The 3990 cache acts as a secondary buffer. It is not useful to store the same data in processor storage and the 3990 cache. To be useful, the 3990 cache must be significantly larger than the buffers in real storage, store different data, or provide another performance advantage. Data can be retrieved more quickly from 3990 cache than it can be retrieved from DASD. In addition, 3990 cache enables

extended functions such as DASD Fast Write, concurrent copy, and dual copy. You can use DFSMS to provide dynamic management of the cache storage.

If you are using RAMAC DASD, then DASD Fast Write and 3990 caching are always recommended.

How Much 3990 Cache?

The amount of 3990 cache to use for DB2 depends primarily on the relative importance of price and performance. If the 3990 cache is substantially larger than the DB2 buffer pools, DB2 can make effective use of the 3990 cache to reduce I/O times for random I/O. For RAMAC DASD, specify a minimum of 256MB of cache storage and use 3990 Model 6 control units for performance-sensitive work.

For sequential I/O, the improvement the 3990 cache provides is generally small. However, 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 DASD. 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 3990 cache for sequential prefetch. On panel DSNTIPE, you can specify whether to use the sequential mode to read cached data from a 3990 Model 3 or 3990 Model 6 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 3990 cache. List prefetch always bypasses the cache.

Recommendation: If you have a cached 9343, 3990 Model 3 with the extended platform, or 3990 Model 6, specify SEQ for the cache option on panel DSNTIPE. This option can improve performance because 3990 extended platform caching can transfer data between DASD 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 the 3990 cache and degrade system performance. For instance, one large sort could use 100 sequential files, needing 60MB of storage. Unless the 3990 cache sizes are large, you might need to specify BYPASS or use DFSMS controls to prevent the use of the cache during sort processing.

If the DASD is dedicated to sorting and is not RAMAC, you might want to turn off caching on a volume basis. Separate units for sort work can give better performance.

Utility Cache Option

If you are using 3990 caching, and you have the nonpartitioning indexes on RAMAC, consider specifying YES on the UTILITY CACHE OPTION field of installation panel DSNTIPE. This allows DB2 to use sequential prestaging when reading data from RAMAC for the following utilities.

- LOAD PART integer RESUME
- REORG TABLESPACE PART

For these utilities, prefetch reads remain in the cache longer, thus possibly improving performance of subsequent writes.

DASD Fast Write

The DASD Fast Write function of the 3990 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. Combining 3990 cache, including DASD Fast Write, with 3390-9 or RAMAC DASD can permit greater amounts of log data to be stored for less cost. Storing adequate amounts of log data on DASD is crucial for restart and recovery performance.

Dual Copy

Dual copy, a 3990 extended function, can be useful for ensuring high availability of some key DB2 data objects, most typically the DB2 catalog, directory, and some key, large secondary indexes. If the DB2 catalog and directory are damaged, all of the other data cannot be accessed. Some secondary indexes on very large, partitioned tables can require dual copy to meet the service-level agreement for recovery time.

Concurrent Copy

DB2 can also benefit from the improvements in copy provided in the 3990 extended platform. See Section 2 of *Utility Guide and Reference* for information about using DFSMS Concurrent Copy to obtain consistent point-in-time concurrent backups of DB2 data.

Direct-access Storage Devices (DASD)

DASD offers direct access to data at high speed and with high data transfer rates, providing an excellent combination of price and performance among the various permanent storage media. Within the range of DASD products, the 3990-6 storage controller combined with RAMAC DASD offers the best performance for most work loads.

Optical Storage

Optical storage complements DASD and supplements tape by providing low-cost, direct access to online information. As a storage alternative, it can cost less than DASD, and, like tape, it is a mountable medium. Optical storage is not necessarily a replacement for magnetic storage, but it can be cost effective in the following cases:

- *The application can tolerate longer response times.* With optical storage, I/O operations are significantly slower than with magnetic devices. The slower response times could affect direct response times, indirect response elongation, and batch windows. The impact of optical storage's slower I/O response time varies. In cases where processor time dominates or I/O overlaps, the difference is negligible. In other cases, the slower I/O operations result in longer response times.
- *The application needs large amounts of storage.* With optical storage, the cost per byte is only a fraction of that for standard DASD. For applications that depend on tens or hundreds of gigabytes of data, optical storage is a feasible option. In some cases, the lower storage costs allow data to be retained on line rather than deleted or moved to tape.

- *The application's data is not accessed frequently.* The key to achieving acceptable performance with optical storage is to avoid mounts when possible. The media used in optical storage devices is not permanently mounted in the drives. If you access data on optical storage too frequently, contention for the optical drives and excessive mount activity can occur, leading to extended response times. Normally, you should store data that is accessed more frequently on magnetic DASD.

Tape or Cartridge

Magnetic tape or cartridge storage, which offers sequential access to data, supports data residing on DASD by providing back-up, recovery, and archive functions. It is also appropriate for very large data sets that do not need to be on line. Tape is a storage option for DB2 image copies and archive logs, for example. Although DB2 databases cannot be directly stored on tape, you can use DFSMShsm to migrate and recall DB2 data from tape.

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” in MVS Version 5 Release 1 or later.

- Using goal mode, for MVS Version 5 Release 1 or later.

  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. If you have MVS Version 5 Release 2 or later, you can also set response time goals for distributed users, as described in “Using Workload Manager to Set Performance Objectives” on page 5-124. For more information about setting response time goals for users, see *MVS/ESA Planning: Workload Management* .

  Setting response time goals does require certain levels of software for the various transaction managers:

  – CICS Version 4 Release 1

  – IMS Version 5 Release 1

  – For DDF threads, a combination of DB2 Version 4 Release 1 and MVS Version 5 Release 2 for DDF threads

  – For threads using stored procedures, a combination of DB2 Version 5 and OS/390 Release 3.

For DB2 address spaces, **velocity goals** are more appropriate, and velocity goals are what we focus on in this section. 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 two ways to set DB2 address space performance options:

- "Using SRM (Compatibility Mode)"
- "Using MVS Workload Management Velocity Goals" on page 5-111

Using SRM (Compatibility Mode)

You can run in compatibility mode in MVS Version 5 or subsequent releases 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 *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 address space
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)
5. DB2 database services address space (*ssnm*DBM1)

   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.
6. DB2-established stored procedures address space (*ssnm*SPAS) and any WLM-established stored procedures address spaces.

   Stored procedures address spaces are special allied address spaces that allow DB2 to load and execute stored procedures in an environment that is isolated from the DB2 subsystem. For a DB2-established stored procedures address space, multiple stored procedures run in the single address space. The address space has the ability to manage work queues associated with each available stored procedure name. This should be set similarly to the application driving the stored procedure.

   For WLM-established stored procedures address spaces, of which there can be many, the work of the stored procedures is managed at the priority of the stored procedures caller (IMS or CICS, for example). The priority you set here for the WLM-established address spaces is used only for system work that is not part of the stored procedure itself.
7. CICS application owning regions
8. IMS dependent regions or TSO address spaces
9. Distributed data facility address space (*ssnm*DIST)

   Although we have the DDF address space listed last here, where it goes depends on which release of MVS you are running. With MVS Version 5 Release 1 and earlier releases, give the DDF address space a priority that is similar to other address spaces performing similar work. For example, if you

use DDF mainly for query work, assign a priority that is similar to the priority used for TSO QMF users. If DDF is used for light transaction work, assign a priority higher than TSO, perhaps similar to IMS dependent regions.

With MVS Version 5 Release 2 and subsequent releases, DDF threads run at the priority of the caller, so we recommend that you place the DDF address space a higher priority, perhaps similar to *ssnm*DBM1.

I/O Scheduling Priority

If your DB2 subsystem is running with MVS/SP 4.3.0 or later and DFSMS/MVS 1.1 or later, DB2 can schedule synchronous read/write I/Os and prefetch read I/Os under the application address space's I/O scheduling priority. To do this, you must do both of the following:

- Enable MVS I/O priority scheduling by specifying IOQ=PRTY in the IEAIPSxx 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.

When you do this, the following scheduling priorities are in effect:

- The application's address space determines the I/O scheduling priority for single-page synchronous read and write I/Os, sequential prefetch, list sequential prefetch, and sequential detection.
- DB2's database services address space (*ssnm*DBM1) determines the I/O scheduling priority for asynchronous write I/Os.

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.

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.

Also be sure that *ssnm*DBM1 has a higher priority than TSO and batch. This helps ensure that deferred write I/Os are scheduled before prefetch read I/Os, thereby preventing a shortage of available buffers.

For more information on the IOP and IOQ parameters, see *MVS/ESA Initialization and Tuning Reference*.

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, it should also be provided for the DB2 address spaces and the IRLM.

Work Load Control

Performance groups and performance-group periods can be used effectively to prioritize the TSO, batch, and QMF 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.

Using 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. You should 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.

In this section, we give recommendations for 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, DDF, or stored procedures 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 address space
  IRLM address space (IRLMPROC)

- 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 system services address space (*ssnm*MSTR)
  DB2 database services address space (*ssnm*DBM1)
  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

- If your system is running MVS/ESA Version 5 Release 2 or a subsequent release, set the DB2 distributed data address space (*ssnm*DIST) in the same service class as *ssnm*DBM1. If you are running a release of MVS before Version 5 Release 2, 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. It is best to run *ssnm*DIST in a service class with other started tasks with similar business goals, such as PRODREGN or some other medium-level service class.

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 we recommend the following service classes for velocity:

- The default SYSSTC service class for:

  VTAM address space
  IRLM address space (IRLMPROC)

- 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)

When running with CICS Version 4 Release 1 or subsequent releases, or with IMS Version 5 Release 1 or subsequent releases, both of which use workload manager, 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 5-124.

For the DB2-established stored procedures address space, use a 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 run as a started task to make it 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.
- In releases of MVS before OS/390 Release 3, workload manager in goal mode uses the processor dispatching priority to set the I/O priority. With OS/390 Release 3, and subsequent releases, 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 specificy in the IPS PARMLIB member. Goal mode does *not* use the IPS PARMLIB member.

  If you are determining I/O priority for DB2 read I/Os, be aware of the relationship between the type of work DB2 is doing and how the I/O priority is determined. See Table 68.

Table 68. How I/O Dispatching Priority is Determined

Request Type	Read I/O Priority is determined by...
Not DDF	Application's address space
DDF	DDF's address space
Parallel task on assisting DB2 (Sysplex query parallelism only)	*ssnm*DBM1 address space

- MVS workload management dynamically manages storage isolation to meet the goals you set.

Chapter 5-6. 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 discusses the following:

- 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 5-116.
- 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 5-121.
- Design options for reducing thread allocations and improving performance generally, under "CICS Design Options" on page 5-128, "IMS Design Options" on page 5-134, and "TSO Design Options" on page 5-135.

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 Section 2 of *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 5-121.

Step 1: Thread Creation

If there is no existing thread that can be reused, the first execution of an SQL statement causes a thread to be created. During thread creation using ACQUIRE(ALLOCATE), the resources needed to execute the application are acquired. During thread creation using ACQUIRE(USE), only the thread is created.

The list below 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. This plan is bound with ACQUIRE(USE), so the trace does not show the descriptors loaded; 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 5-120.

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. If most of the SQL is used in every execution of the transaction, 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 5-66 for more information.

Step 2: Resource Allocation

When a thread has been created, DB2 processes the first SQL statement executed.

Some of the structures necessary to process the statement are stored in 4KB 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

When plan allocation ends, the statement is executed. 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 3-27.

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 5-4. Tuning DB2 Buffer, EDM, RID, and Sort Pools" on page 5-49.

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 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 4-109.

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 5-90 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, this is when table space locks are released, the DBD use count is decreased, and the thread storage is released.

Variations on Thread Management

There are minor differences in the transaction flow in different environments and when the SQL statement originates dynamically.

TSO and Call Attachment Facility Differences

The TSO attachment facility and call attachment facility (CAF) can be used to request that SQL statements 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 Section 6 of *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: Managing QMF for MVS*.

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:

- "Thread Reuse for Database Access Threads" on page 5-124 provides information on the conditions for thread reuse for database access threads.
- "CICS Design Options" on page 5-128 tells how to write CICS transactions to reuse threads.
- "IMS Design Options" on page 5-134 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 temporary tables, the logical work file space is not released until the thread is deallocated. Thus, many uses of the same 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 107 on page 5-121, 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 107. DB2 PM Accounting Report - Information about Thread Termination

Database Access Threads

This section describes:

For information on performance considerations for distributed processing, see "Considerations for Tuning Distributed Applications" on page 5-316.

Differences Between 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 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 SRB mode.
- For database access threads, there is no create thread stage. A database access thread persists until the connection between the two systems terminates.
- 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.

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 25 000 as the maximum number of active and inactive database access threads that can concurrently exist within DB2. This upper limit is only obtainable if you specify the recommended value INACTIVE for the DDF THREADS field of installation panel DSNTIPR.

| For TCP/IP connections, it is a good idea to specify the IDLE THREAD TIMEOUT
| value in conjunction with a TCP/IP keep_alive timer of 5 minutes or less to make
| sure that resources aren't locked for a long time when a network outage occurs.

Comparing Active and Inactive Threads

A database access thread that does not hold any cursors or database resources
(such as storage) is known as an *inactive thread*. To allow a thread to become
inactive (what might be called a 'sometimes active' thread), the following conditions
must be true:

- The DDF THREADS field of installation panel DSNTIPR must contain
INACTIVE.
- The package must have been bound with RELEASE(COMMIT).
- There must be no open cursors defined WITH HOLD.

When the above conditions are true, the thread can become inactive when a
COMMIT is issued. A ROLLBACK makes a thread become inactive even if there
are open cursors defined WITH HOLD because ROLLBACK closes all cursors. The
advantages of sometimes active threads are:

- 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,
thus avoiding the delay of repeated connections.
- DB2 can support a larger number of DDF threads (25 000 instead of 2000).
- Less storage is used for each DDF 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 5-126 for more information.

The response times reported by RMF do not include inactive periods between requests.

- 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."

Accounting for Inactive Threads

The accounting trace record (IFCID 0003) is produced every time a thread becomes inactive. You can see this count in the DBAT INACTIVE field of the "Application Termination" section of the DB2 PM accounting report. (See Figure 107 on page 5-121.)

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 indoubt thread. A value of 0, the default, means that the server threads cannot be canceled because of an idle thread timeout.

We recommend that this option be used 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.

How a Database Access Thread Is Created

The following steps occur during the creation of 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 option on installation panel DSNTIPR, database access threads can be active or inactive, depending on the limits you specified for the MAX REMOTE CONNECTED and MAX REMOTE ACTIVE fields of panel DSNTIPE. 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 conversation is deallocated.

   If the MAX REMOTE CONNECTED limit has not been reached, the thread creation 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 thread request until the MAX REMOTE ACTIVE value falls below the limit. Until that happens, this thread is an inactive thread. When the number of active threads falls below this

MAX REMOTE ACTIVE limit, the queued inactive thread completes the thread creation process and becomes an active thread.

During this waiting period, you can check the status of the active threads using the command DISPLAY THREAD(*) TYPE(ACTIVE). You can check the status of inactive threads by using DISPLAY THREAD(*) TYPE(INACTIVE).

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.

Thread Reuse for Database Access Threads

Only DB2 for OS/390 requesters support thread reuse. However, in the case of a DB2 requester connected to a server DBMS other than DB2 for OS/390, thread reuse causes the connection to be released and connected again using the new authorization ID. As a result, there are no performance advantages for thread reuse in connections to a server other than DB2 for OS/390.

Using Workload Manager to Set Performance Objectives

MVS/ESA Version 5 Release 2 and subsequent releases support 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 *MVS/ESA 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 5-108, 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 5-127.

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. Here are WLM classification attributes that pertain to DB2 DDF threads:

AI Accounting information. The value of the DB2 accounting string associated with the DDF server thread. This is described by QMDAAINF in the DSNDQMDA mapping macro.

CI The DB2 correlation ID of the DDF server thread. This is 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.

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, you would 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.

UI User ID. The DDF server thread's primary authorization ID, after inbound name translation.

Figure 108 on page 5-126 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 108. Classifying DDF Threads using Workload Manager. You assign performance goals to service classes using the services classes menu of WLM.

In Figure 108, the following classifications are shown:

- All DB2P applications in COLLECTION ONLINE are in the PRDONLIN service class.
- The stored procedure PAYPROC is in service class PRDONLIN.
- All work performed by DB2P user SYSADM is prioritized as online work.
- 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 prioritized as batch work.
- All users of the test DB2 system are assigned to the TESTUSER class except for the PAYPROCT stored procedure, which is in a test stored procedures class.

Don't Create Too Many Stored Procedures Address Spaces: Workload manager creates one stored procedures address space for every combination of service class and WLM environment name for which work exists. See “Assigning Stored Procedures to WLM Application Environments” on page 5-329 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. So, 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 5-326.

Because threads that are always active do not terminate the MVS enclave, we recommended that you do not use multiple performance periods for always active threads because the last period performance objective is used to manage most of the work done by the thread.

Basic Procedure for Establishing Performance Objectives

To establish performance objectives for DDF threads and the related address spaces:

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 serivce 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: When you run in compatibility mode, you have to take on more performance management issues. With stored procedures that run in WLM-established address spaces, for example, WLM cannot automatically start new stored procedures address space to handle additional high-priority stored procedures requests, as it can when using goal mode. You must monitor the

performance of the stored procedures to determine how many stored procedures 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."

The DDF address space and any WLM-established stored procedures address spaces should be assigned to the same service class as the DB2 database services address space (*ssnm*DBM1). This service class should be defined with a velocity goal.

You can assign 'sometimes active' threads to any goal type and number of periods. For always active threads, on the other hand, use velocity goals or discretionary goals and use a single period service class. This is because the response times for an always active thread are unrelated to the amount, duration, or resource consumption of the underlying work.

Stored Procedures: When you are in goal mode, WLM automatically starts WLM-established stored procedures address spaces to help meet the service class goals you set. This is assuming you have defined the application environment, as described in "Assigning Stored Procedures to WLM Application Environments" on page 5-329.

CICS Design Options

The information under this heading, up to "IMS Design Options" on page 5-134, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page xi.

This section describes the following:

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 “Correlating Accounting Information for CICS Threads” on page 5-134.

For more information about specifying the CICS attachment facility macros, see Section 2 of *Installation Guide*.

Managing Plans for CICS Applications

You can use either packages or dynamic plan selection to manage your CICS applications, but packages offer more flexibility. See *Application Programming and SQL Guide* for more information about using packages. See “Routines for Dynamic Plan Selection in CICS” on page X-71 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 in order to increase the chances of thread reuse. That time is determined by the purge cycle, normally 30 seconds.

States of Threads: We use the following terms to 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 68 on page 4-47 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. There are controls in CICS to 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 5-134 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.

When Sign-on Occurs: 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 “Correlating Accounting Information for CICS Threads” on page 5-134.
- The last transaction left a held cursor open.
- The last transaction left one of the modifiable special registers in use.

Using TXIDSO to Control Sign-on Processing: With CICS Version 4 or later, 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 so, go to the next step. If not, the thread cannot be released until EOT.

  2. Are the following special registers in their initial state?

     CURRENT SQLID
     CURRENT SERVER
     CURRENT PACKAGESET
     CURRENT RULES
     # CURRENT PRECISION

     If yes, go to the next step. If 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. Are all cursors declared WITH HOLD closed?

     If yes, this thread can be released at SYNCPOINT. If no, and this is a local connection, this thread cannot be released until EOT.

     If 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 yes, the thread can be released. If not, 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. For the remainder of the time that CICS is up, it uses the following, depending on which version of the CICS attachment you are using:

- For the attachment used with CICS release prior to CICS Version 4, the purge cycle is 30 seconds
- 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.

When TCBs are Detached: 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

We recommend that you first set the RCT parameters as follows:

- 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 5-134.

  - 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.

Do not misread this situation and then set DPMODE=HIGH, because the problem will then 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). With DPMODE=EQUAL, the thread TCBs actually have an MVS dispatching priority slightly lower than the CICS main task TCB.

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 low-priority, short SQL transactions, especially non-terminal-driven transactions.

Recommendations for CICS System Definitions

The following specifications control how many tasks can be created for a transaction class:

- Maximum task class specification (CMXT) in the SIT (for CICS 3.3 and earlier releases)
- CEDA DEFINE TRANCLASS() GROUP() MAXACTIVE() in the CSD (for CICS Version 4 and later)

Recommendations for setting CMXT or MAXACTIVE are:

- When TWAIT=YES and there are unprotected threads, use the value of THRDA plus one.
- When TWAIT=POOL, then 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.

Correlating 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

Using the IMS attachment facility, you can:

- 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 which can connect to DB2 and by transaction class assignments. We recommend that you:

  - 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 109 on page 5-136) 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 109. Thread Queuing in the DB2 PM Statistics Report

QMF Design Options

Some of the significant performance options in QMF are:

- 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: Managing QMF for MVS*.

Chapter 5-7. Improving Concurrency

Before going into detail, this chapter begins by describing:

- "What Is Concurrency? What Are Locks?" on page 5-138,
- "Effects of DB2 Locks" on page 5-139, and
- "Basic Recommendations to Promote Concurrency" on page 5-141.

After the basic recommendations, the chapter tells what you can do about two major techniques that DB2 uses to control concurrency. Those two techniques are *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 5-145 describes the various types of transaction locks that DB2 uses and how they interact.
  - "Tuning Your Use of Locks" on page 5-162 describes what you can change to control locking. Your choices include:
    - "Startup Procedure Options" on page 5-163
    - "Installation Options for Wait Times" on page 5-163
    - "Other Options that Affect Locking" on page 5-168
    - "Bind Options" on page 5-172
    - "Specifying Isolation by SQL Statement" on page 5-184
    - "The Statement LOCK TABLE" on page 5-185

  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, we don't describe them in any detail.

- **Claims and drains** control access by DB2 utilities and commands. An application that accesses an object first makes a claim for it. By the process of draining, a command or utility can quiesce existing claimers and prevent new claims. After the drainer completes its operations, claimers can resume theirs.

  "Controlling Concurrency for Utilities and Commands" on page 5-186 describes how to plan utility jobs and other activities to maximize efficiency.

  We describe how you can monitor DB2's use of locks and show how to analyze a particular sample problem in detail, under the heading:

  "Monitoring DB2 Locking" on page 5-191

DB2 extends its concurrency controls to multiple subsystems for data sharing. For information about that, see *Data Sharing: Planning and Administration*.

What Is Concurrency? What Are 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 generally 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 until 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.

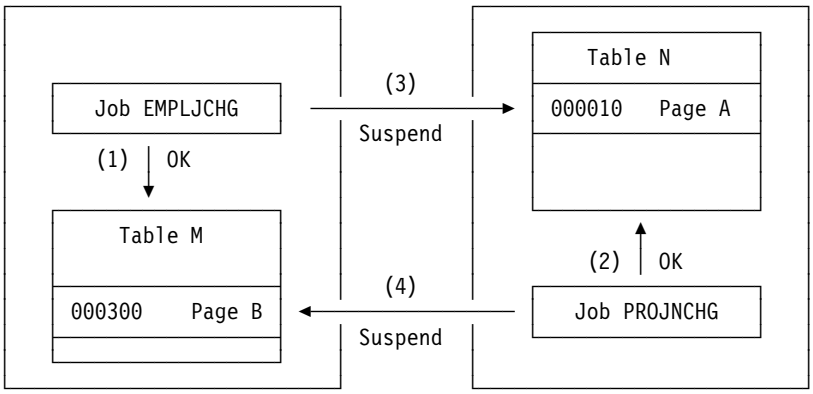
COMMIT and ROLLBACK operations do not time out. The command STOP DATABASE, however, may time out and send messages to the console, but it will retry up to 15 times.

For more information about setting the timeout interval, see “Installation Options for Wait Times” on page 5-163.

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 110 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 110. 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 Section 5 of *Application Programming and SQL Guide*.)

It is possible for two processes to be running on separate 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

The following recommendations are grouped by their scope:

- "Recommendations for System Options"
- "Recommendations for Database Design"
- "Recommendations for Application Design" on page 5-143

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. It should comes 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 the updating of partitioning key columns
when the update moves the row from one partition to the other can cause
concurrency problems. Allow updating only when the row stays in the same
partition by setting the system parameter PARTKEYU 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.

Cluster Your Data: Try to keep data that is accessed together on the same page. A table that starts empty at the beginning of the job and is filled by insertions is not effectively clustered. All of the insertions are at the end of the table and cause conflicts, especially while the table is nearly empty and the index has only one or
| two levels. Type 2 indexes can help alleviate this situation.

For information on using clustering indexes to keep data clustered, see "Clustering Indexes" on page 2-56.

On the other hand, if your application does sequential batch insertions for which the
input data is not in clustering sequence, there can be excessive contention on the
space map page for the table space. 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) for the SQL INSERT statement. For more information about using this option in data sharing, see Chapter 7 of *Data Sharing: Planning and Administration*. For the syntax, see Chapter 6 of *SQL Reference*.

For Changes to Data, Consider Type 2 Indexes: INSERT, UPDATE, or DELETE operations require a lock on every affected page or subpage of a type 1 index, but not on pages of a type 2 index. If there are no type 1 indexes on the data, only the affected data pages or rows are locked. Because there are usually fewer rows to a data page than there are index entries to an index page or subpage, locking only the data when you lock pages likely causes less contention than locking the index. Locking only data rows would likely cause even less contention.

Changes can also split an index leaf page, which locks out concurrent access to a type 1 index but not to a type 2. And if the page has more than one subpage, there can be additional splitting for subpages. Type 2 indexes have no subpages.

If you have had contention problems on index pages, then switch to type 2 indexes. That change alone is likely to solve the problems.

If you insert data with a constantly increasing key, use a type 2 index. The type 1 index splits the last index page in half and adds the new key at the end of the list of entries. It continues to add new keys after that, wasting one-half of the old split page. The type 2 index merely adds the new highest key to the top of a new page, without splitting the page.

Be aware, however, that DB2 treats nulls as the highest value when deciding where to insert values. When the existing high key contains a null value in the first column that differentiates it from the new key that is inserted, then inserted non-null index entries cannot take advantage of this special “highest value” split.

For example, if the existing high key is:

```
SMITH ROBERT J
```

and you insert:

```
SMITH ROBERT (null)
```

Then an insert of:

```
SMITH ROBERT Z
```

is not treated as the new high key.

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. Before you use LOCKSIZE TABLESPACE or LOCKSIZE TABLE, you should know why you do not need concurrent access to the object. 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 type 2 indexes and 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.

| MAXROWS does not apply to the index. If you are still using type 1 indexes, you
| can pad the index by adding a long column to the index. However, this technique
| does not force each index entry to a separate page or subpage.

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 REORD, to break in

Consider using the UR CHECK FREQ field of the installation panel DSNTIPN to
help you identify those applications that are not committing frequently. The setting
of UR CHECK FREQ should conform to your installation standards for applications
taking commit points.

Even though an application may be conforming to the commit frequency standards of the installation under normal frequency standards of the installation under normal operational conditions, variation can occur based on system workload fluctuations. For example, a low-priority application may issue a commit frequently on a system that is lightly locaded. 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. That 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 release those locks and free the resources they hold.

Bind Plans with ACQUIRE(USE): That choice is best for concurrency. Packages are always bound with ACQUIRE(USE), by default. ACQUIRE(ALLOCATE) gives better protection against deadlocks for a high-priority job; if you need that option, 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 locks as soon as possible. CURRENTDATA(NO) lets DB2 avoid acquiring 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 you have accessed must not be changed before your next FETCH operation.
2. ISOLATION(RS), when rows you have accessed 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 rows you have accessed must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.

Use ISOLATION(UR) Cautiously: UR isolation acquires almost no locks. 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.

Aspects of Transaction Locks

Four Basic Aspects:

- "The Size of a Lock"
- "The Duration of a Lock" on page 5-148
- "The Mode of a Lock" on page 5-149
- "The Object of a Lock" on page 5-152

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:

- "What Lock Types DB2 Chooses" on page 5-154

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.

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 111 on page 5-146 suggests, row locks and page locks occupy an equal place in the hierarchy of lock sizes.

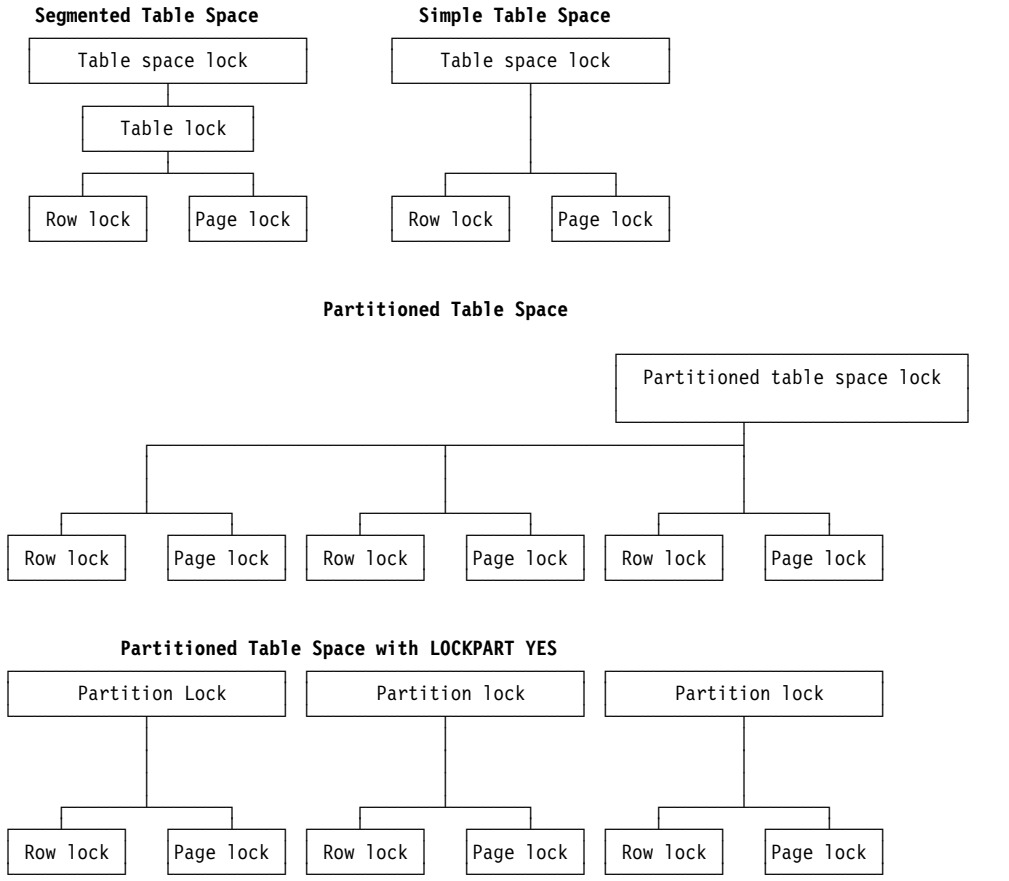


Figure 111. Sizes of Objects Locked

General Effects of Size: Locking larger or smaller amounts of data allows you to trade performance for concurrency. When you use page or row locks instead of table or table space locks:

- 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.

When you use only table or table space locks:

- 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 on 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 7 of *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 are not affected by update activity.

Restrictions: If any of the following conditions are true, DB2 must lock *all* partitions when LOCKPART YES is used:

- A type 1 index is used in the access path
- The plan is bound with ACQUIRE(ALLOCATE)
- The table space is defined with LOCKSIZE TABLESPACE
- When LOCK TABLE IN EXCLUSIVE 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. A single row, of course, contains data from only one table, so 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.

Differences between Simple and Segmented Table Spaces: Figure 112 on page 5-148 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. Even a single page can contain rows from both T1 and T2. If User 1 and User 2 acquire locks on different pages, neither can access all the rows in T1 and T2 until one of the locks is released.

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.

For information about controlling the size of locks, see:

- “LOCKSIZE Clause of CREATE and ALTER TABLESPACE” on page 5-169
- “The Statement LOCK TABLE” on page 5-185

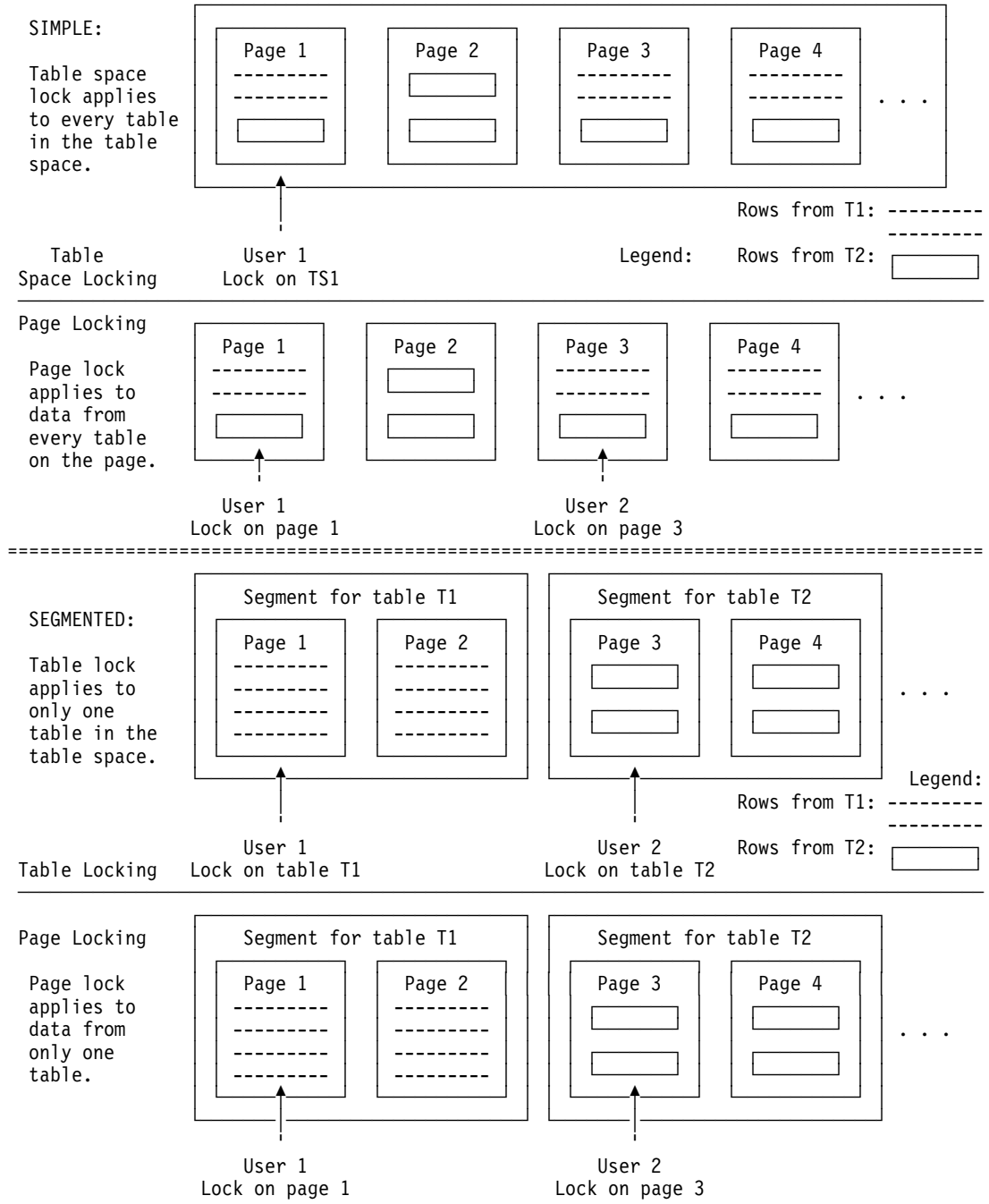


Figure 112. Page Locking for Simple and Segmented Table Spaces

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.

Example: An application locates customers in a table of customer data and changes their addresses. The statement locks the entire table space and the specific rows that it changes. The application might acquire the lock on the table space as soon as the program starts and hold the lock until the program ends. It would acquire the lock on a specific row only when it accesses the row and can release the row lock when it commits the change to that row.

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.

| To use selective partition locking (LOCKPART YES), you must bind the plan to acquire the locks at the first use of the resource (the BIND option is ACQUIRE(USE). All locks on partitions of a particular table space acquired by a particular application are held for the same duration.

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 5-172.

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.

Figure 113 on page 5-150 lists the possible modes for page and row locks;
| Figure 114 on page 5-151 lists the modes for partition, table, and table space locks.

| 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. Processes concurrent with the U lock can acquire S locks and can read the page or row, 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.)

Figure 113. Modes of Page and Row Locks

Effect of the Lock Mode: 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.)

Which Lock Modes are Compatible? Compatibility for page and row locks is easy to define: Table 69 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 69. 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

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.

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. The lock owner does not need page or row locks.

Figure 114. Modes of Table, Partition, and Table Space Locks

Compatibility for table space locks is slightly more complex. Table 70 on page 5-152 shows whether or not table space locks of any two modes are compatible.

Table 70. 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: The *object* of a lock is the resource being locked.

Examples

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 indexes**. DB2 acquires locks on pages and subpages of type 1 indexes. An advantage of type 2 indexes is that they are protected by locks on the underlying data pages or rows; the index pages themselves are not locked. For more information, see "Locks on Indexes," below.
- **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.
- **DB2 internal objects**. Most of those you are never aware of. Some locks you might notice are on:
  - Portions of the **DB2 catalog**. For more information, see "Locks on the DB2 Catalog" on page 5-153.
  - 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 5-154.
  - The **database descriptor** (DBD) representing a DB2 database. For more information, see "Locks on the Database Descriptors (DBDs)" on page 5-154.

Locks on Indexes

Type 1 Indexes: A lock on a table or table space also protects every page or subpage of every type 1 index on the table space, as though each one was locked with the same mode. If DB2 acquires page locks in the table space, it might also acquire locks on pages or subpages of the indexes. (It cannot acquire row locks if the table space has any type 1 indexes.)

Thus, if an application process has an S or X lock on a table space, the indexes are inherently locked in S or X mode, and index pages or subpages are not locked

separately. If the process has an IS, IX, or SIX lock on the table space, particular index pages or subpages can be locked separately.

Type 1 Index Disadvantage: A single process accessing data through a type 1 index can sometimes experience a deadlock between a data page and an index page.

Type 2 Indexes: The transaction lock acquired on a table space protects all type 2 indexes on all tables in the table space. If the process changes an index key, only the data that the key points to is locked. That technique, called *data-only locking*, greatly reduces contention for index pages. Transactions that insert high key values at the end of an index benefit greatly.

Type 2 Index Disadvantage: A query that uses index-only access might lock the data page or row if the index is type 2, and that lock can contend with other processes that lock the data. Index-only access with a type 1 index does not acquire any data page or row locks.

Locks on the DB2 Catalog

SQL data definition statements, GRANT statements, and REVOKE statements require locks on the DB2 catalog. The catalog is designed to minimize contention among application processes, but there are also active steps you can take 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: To reduce contention, convert the indexes on the catalog and directory to type 2, and 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.
- If a CREATE statement causes a page of a type 1 index to split, it limits concurrency by locking more than one index page. That happens most often when DB2 is first installed. We recommend converting all indexes on the DB2 catalog and directory to type 2.
- 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 71.

Table 71. 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

What Lock Types DB2 Chooses

Overview: For the locks acquired on target tables and indexes by different types of SQL data manipulation statements, see:

- “Modes of Locks Acquired for SQL Statements” on page 5-155, below

But 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 5-159
- “Lock Escalation” on page 5-160

For a summary of the locks acquired by other operations, see:

- “Modes of Transaction Locks for Various Processes” on page 5-161

Modes of Locks Acquired for SQL Statements

Table 72 on page 5-156 shows the modes of locks that a process acquires. The mode depends on:

1. The type of processing being done
2. The value of LOCKSIZE for the target table
3. The value of ISOLATION with which the plan or package is bound
4. The method of access to data

For details about:

- LOCKSIZE, see “LOCKSIZE Clause of CREATE and ALTER TABLESPACE” on page 5-169
- ISOLATION, see “The ISOLATION Option” on page 5-176
- Access methods, see “Chapter 5-10. Using EXPLAIN to Improve SQL Performance” on page 5-261

Reading the Table: We illustrate the steps with the following example:

```
EXEC SQL DELETE FROM DSN8510.EMP WHERE CURRENT OF C1;
```

Your steps are:

1. Find the section of the table for DELETE operations using a cursor. It is on page 5-158.
2. Find the row for the appropriate values of LOCKSIZE and ISOLATION. Table space DSN8510 is defined with LOCKSIZE ANY. If the value of ISOLATION was not specifically chosen, it is RR by default. You are interested in the 4th row of this section of the table.
3. Find the subrow for the expected access method. We can suppose the operation uses the index on employee number, and for the example we assume that is a type 1 index. Because the operation deletes a row, it must update the index. Hence, you can read the locks acquired in the subrow for “Type 1 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
   - An X lock on the index page (but only if that is a type 1 index)
4. Check the notes to the entries you use, at the end of the table. For this sample operation, see:
   - Note 3, on the column heading for “Table.” If the table is not segmented, there is no separate lock on the table.
   - Note 4, 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. And because we assume that the index is type 1, the table space cannot have row locks.

Table 72 (Page 1 of 4). Modes of Locks Acquired for SQL Statements. Numbers in parentheses () refer to numbered notes on page 5-159.

			Lock Mode			
LOCKSIZE (1)	**ISO-LATION**	**Access Method (2)**	**Table Space (11)**	**Table (3)**	**Data Page or Row (4)**	**Index Page**
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	n/a
TABLE (3)	CS RS RR	Any	IS	S	n/a	n/a
PAGE, ROW, or ANY	CS or RS	Type 1 probe or scan	IS(5)	IS(5)	n/a	S
		Type 1 data retrieval	IS(5)	IS(5)	S(6)	S
		Type 2, any use	IS(5) (12)	IS(5)	S(6)	n/a
		Table space scan	IS(5) (12)	IS(5)	S(6)	n/a
PAGE, ROW, or ANY	RR	Type 1 probe	IS(5)	IS(5)	n/a	S
		Type 1 data retrieval	IS(5)	IS(5)	S	S
		Type 1 index scan (7)	IS(3) or S	S, IS, or n/a	n/a	S or n/a
		Type 2 probe or data retrieval	IS(5)	IS(5)	S	n/a
		Type 2 scan (7)	IS(3) or S	S, IS, or n/a	S or n/a	n/a
		Table space scan (7)	IS(3) or S	S or n/a	n/a	n/a
Processing statement: INSERT ... VALUES(...) or INSERT ... subselect (8)						
TABLESPACE	CS RS RR	Any	X	n/a	n/a	n/a
TABLE (3)	CS RS RR	Any	IX	X	n/a	n/a
PAGE, ROW, or ANY	CS RS RR	Type 1, any use	IX	IX	X	X
		Type 2, any use	IX	IX	X	n/a
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	n/a
TABLE (3)	CS RS RR	Any	IX	X	n/a	n/a
PAGE, ROW, or ANY (13)	CS	Type 1 index selection	IX	IX	X	U→X
		Type 1 data selection	IX	IX	U→X	U→X
		Type 2 index selecton	IX	IX	X (delete) U→X (update)	n/a
		Type 2 data selection	IX	IX	U→X	n/a
		Table space scan	IX	IX	U→X	n/a

Table 72 (Page 2 of 4). Modes of Locks Acquired for SQL Statements. Numbers in parentheses () refer to numbered notes on page 5-159.

			Lock Mode			
LOCKSIZE (1)	**ISO-LATION**	**Access Method (2)**	**Table Space (11)**	**Table (3)**	**Data Page or Row (4)**	**Index Page**
PAGE, ROW, or ANY (13)	RS	Type 1 index selection	IX	IX	X	S or U(10)→X
		Type 1 data selection	IX	IX	S or U(10)→X	S or U(10)→X
		Type 2 index selection	IX	IX	S or U(10)→X (update) S→X(10) or X (delete)	n/a
		Type 2 data selection	IX	IX	S or U(10)→X	n/a
		Table space scan	IX	IX	S or U(10)→X	n/a
PAGE, ROW, or ANY	RR	Type 1 index selection	IX	IX	X	S or U(10)→X
		Type 1 data selection	IX	IX	S or U(10)→X	S or U(10)→X
		Type 2 index selection	IX	IX	S or U(10)→X (update) X or S(10)→X (delete)	n/a
		Type 2 data selection	IX	IX	S or U(10)→X	n/a
		Table space scan	IX(3) or X	X or n/a	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	n/a
TABLE (3)	CS RS RR	Any	IS or IX	U	n/a	n/a
PAGE, ROW, or ANY	CS	Type 1 probe or scan	IX	IX	n/a	S or U(9)
		Type 1 data retrieval	IX	IX	U	S or U(9)
		Type 2 index, any use	IX	IX	U	n/a
		Table space scan	IX	IX	U	n/a

Table 72 (Page 3 of 4). Modes of Locks Acquired for SQL Statements. Numbers in parentheses () refer to numbered notes on page 5-159.

			Lock Mode			
LOCKSIZE (1)	**ISO-LATION**	**Access Method (2)**	**Table Space (11)**	**Table (3)**	**Data Page or Row (4)**	**Index Page**
PAGE, ROW, or ANY	RS	Type 1 probe or scan	IX	IX	n/a	S, U, or X(10)
		Type 1 data retrieval	IX	IX	S, U, or X(10)	S, U, or X(10)
		Type 2 index, any use	IX	IX	S, U, or X(10)	n/a
		Table space scan	IX	IX	S, U, or X(10)	n/a
PAGE, ROW, or ANY	RR	Type 1 probe	IX	IX	n/a	S, U, or X(10)
		Type 1 data retrieval	IX	IX	S, U, or X(10)	S, U, or X(10)
		Type 1 scan (7)	IX(3) or X	X, IX, or n/a	n/a	S, U(10), X, or n/a
		Type 2 probe or data retrieval	IX	IX	S, U, or X(10)	n/a
		Type 2 scan (7)	IX(3) or X	X, IX, or n/a	S, U, X(10), or n/a	n/a
		Table space scan (7)	IX(3) or X	X or n/a	n/a	n/a
Processing Statement:		**UPDATE or DELETE with cursor**				
TABLESPACE	Any	Any	X	n/a	n/a	n/a
TABLE (3)	Any	Any	IX	X	n/a	n/a
PAGE, ROW, or ANY	CS, RS, or RR	Type 1 index, updated	IX	IX	X	X
		Type 2 index, updated	IX	IX	X	n/a
		Index not updated	IX	IX	X	n/a

Table 72 (Page 4 of 4). Modes of Locks Acquired for SQL Statements. Numbers in parentheses () refer to numbered notes on page 5-159.

			Lock Mode			
LOCKSIZE (1)	**ISO-LATION**	**Access Method (2)**	**Table Space (11)**	**Table (3)**	**Data Page or Row (4)**	**Index Page**

Notes:

1. If the database is defined with ROSHARE READER, DB2 ignores LOCKSIZE and acquires an S lock on the table space.
2. Access methods are:

Probe	The index alone is searched to find one or more entries.
Scan	The index or table space is scanned for successive entries or rows.
Data retrieval	Data rows are retrieved through index access.
Index selection	The index alone identifies qualifying rows.
Data selection	The index and data are both examined to identify qualifying rows.

   Type 1 and type 2 refer to types of indexes.
3. Used for segmented table spaces only.
4. 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.
5. 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.
6. 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.
7. Even if LOCKMAX is 0, the bind process can promote the lock size to TABLE or TABLESPACE. If that occurs, SQLCODE +806 is issued.
8. 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.
9. The U lock is taken if index columns are updated.
10. 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 5-171. 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.
11. Includes partition locks, if selective partition locking is used.
12. If the table space is defined with LOCKPART YES, it is possible that locks can be avoided on the partitions.
13.

With LOCKSIZE ANY and LOCKMAX=0, the bind process uses a lock size of table space or table.

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: for page or row locks, from S or U to X; for table, partition, or table space locks, from IS to either IX or S, or directly to U, SIX, or X.

Lock Escalation

Definition: *Lock escalation* is the act of releasing a large number of page or row 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.

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.

Lock counts are always kept on a table or table space level, not on a partition level. For a table space defined with LOCKPART YES, lock escalation only occurs for partitions that are currently locked. Unlocked partitions remain unlocked. This means that for a partitioned table space defined with LOCKPART YES, different partitions can be locked with different lock modes. After lock escalation occurs, any unlocked partitions that are subsequently accessed are locked with a gross lock.

Example: If a table space is defined with LOCKSIZE ANY and LOCKMAX 2000, DB2 can use page locks for a process that accesses the table space and can escalate those locks. If the process attempts to lock more than 2000 pages in the table space at one time, DB2 promotes its intent locks on the table space or currently locked partitions to mode S or X and then releases its page locks.

If the process is using Sysplex query parallelism, and each member has a LOCKMAX value of 2000, escalation does not occur until 2000 locks are held on any one of the members running tasks for that query.

When It Occurs: Lock escalation balances concurrency with performance by using page or row locks while a process accesses relatively few pages or rows, 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 5-169
- “LOCKMAX Clause of CREATE and ALTER TABLESPACE” on page 5-170

Lock escalation is suspended during the execution of SQL statements for ALTER, CREATE, DROP, GRANT, and REVOKE.

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 so decrease the number of escalations. Alternatively, you might let the process that is causing them begin by locking the entire table space, using the statement LOCK TABLE. That prevents concurrency, but it is a reasonable solution for some end-of-month or end-of-year situations when a process updates more pages than it normally does.

Example: Assume that a table space is used by transactions that require high concurrency. There is also a batch job that 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 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. Also, bind with ISOLATION(CS), described under "The ISOLATION Option" on page 5-176, to avoid X locks on the table space when making updates.
- *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 5-185.

Modes of Transaction Locks for Various Processes

The rows in Table 73 on page 5-162 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 73. 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
| **Note:** Altering the LOCKPART clause requires an X lock on the table space.				
SQL DROP TABLESPACE statement	IX	X (8)	X	X
SQL GRANT statement	IX	n/a	n/a	n/a
SQL REVOKE statement	IX	X (8)	n/a	n/a

Notes:

1. In a lock trace, these locks usually appear as locks on the DBD.
2. The target table space is the table space:
   - Accessed and locked by an application process,
   - Processed by a utility, or
   - 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 72 on page 5-156.
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.

Tuning Your Use of Locks

This section describes what you can change to affect transaction locks, under:

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*.

Option List

The options relevant to DB2 locking are:

SCOPE Whether IRLM is used for data sharing (GLOBAL) or not (LOCAL). We recommend LOCAL unless you are using 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.

These fields of the installation panels are relevant to transaction locks:

- "DEADLOCK TIME on Installation Panel DSNTIPJ" on page 5-164
- "RESOURCE TIMEOUT on Installation Panel DSNTIPI" on page 5-164
- "IDLE THREAD TIMEOUT on Installation Panel DSNTIPR" on page 5-166

This field is relevant to drain locks:

- "UTILITY TIMEOUT on Installation Panel DSNTIPI" on page 5-166

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:*** Detect deadlocks as quickly as possible. Specify a value of 1,
| in most cases.

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

DB2 uses two factors,

- A timeout period and
- An operation multiplier,

in a scanning schedule to determine whether a process waiting for a transaction lock has timed out.

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. The timeout period and timeout multiplier also apply to retained locks in a data sharing environment when the subsystem parameter RETLWAIT is YES. See *Data Sharing: Planning and Administration* for more information about RETLWAIT.

In some cases, you can modify the multiplier value. Table 74 on page 5-165 indicates the multiplier by type of process, and whether you can change it.

Table 74. 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

See “UTILITY TIMEOUT on Installation Panel DSNTIPI” on page 5-166 for information about modifying the utility timeout multiplier.

The timeout period and timeout multiplier also apply to retained locks in a data sharing environment when the subsystem parameter RETLWAIT is YES. See Chapter 3 of *Data Sharing: Planning and Administration* for more information about RETLWAIT.

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 parameters in DSN6SPRM and reassembling DSNTIJUZ:

BMPTOUT The timeout multiplier for IMS BMP connections. A value from 0-254 is acceptable. A zero means use the default (4).

DLITOUT The timeout multiplier for IMS DL/I batch connections. A value from 0-254 is acceptable. A zero means use the default (6).

The Scanning Schedule: Figure 115 on page 5-166 illustrates this 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.

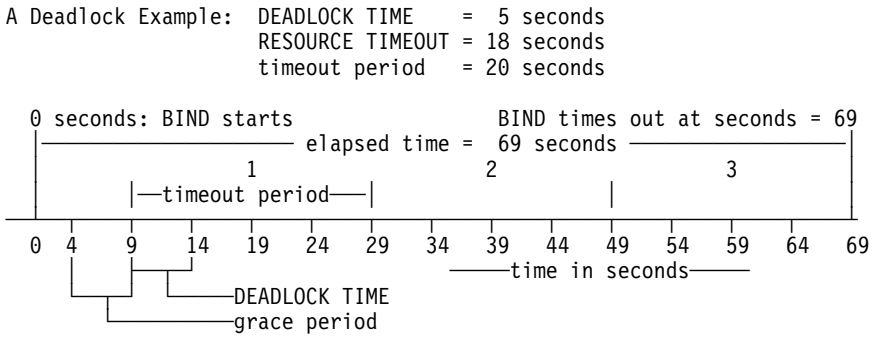


Figure 115. How Long Until Timeout?

Effect: An operation can remain inactive for longer than the value of RESOURCE TIMEOUT.

Recommendation: Consider that effect 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 cancelation 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, as in DB2 Version 3.

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 5-164.

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 75 on page 5-168 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 75. Maximum Drain Wait Times: LOAD Utility

Step	Maximum Wait Time (seconds)
1. Get write drain lock	120
2. Wait for all write 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 repeatable read drain lock	120
6. Wait for all RR claims to be released	120
Total	**720**

Other Options that Affect Locking

There are various options that you can use to control such things as the number of locks that can be gotten and with which mode certain locks are taken. This section describes the following installation options:

- "LOCKS PER USER Field of Installation Panel DSNTIPJ"
- "LOCKSIZE Clause of CREATE and ALTER TABLESPACE" on page 5-169
- "LOCKMAX Clause of CREATE and ALTER TABLESPACE" on page 5-170
- "LOCKS PER TABLE(SPACE) Field of Installation Panel DSNTIPJ" on page 5-171
- "The Option U LOCK FOR RR/RS" on page 5-171
- "Option to Release Locks for Cursors Defined WITH HOLD" on page 5-171
- "Option XLOCK for Searched UPDATEs or DELETEs" on page 5-172

LOCKS PER USER Field of Installation Panel DSNTIPJ

Effect: Specifies the maximum number of page or row locks that can be held by a single process at any one time. It includes locks on data pages, and on index pages and subpages for type 1 indexes, both for the DB2 catalog and directory and for user data.

When a request for a page or row 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: 10 000

Recommendation: We estimate that the default will 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.

Review application processes that require higher values to see if they can use table space locks rather than page or row locks. The accounting trace shows the maximum number of page or row locks a process held while running.

General-use Programming Interface

LOCKSIZE Clause of CREATE and ALTER TABLESPACE

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, or row 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 5-159.

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 5-159.

LOCKSIZE ANY.
DB2 chooses the size of the lock.

For a table space started with read-only access (RO mode), DB2 ignores the LOCKSIZE option and implicitly uses LOCKSIZE TABLESPACE. Thus, the LOCKSIZE clause has no effect for sharers of read-only data. However, if a partition is started with read-only access, and the table space is defined with LOCKPART YES, then an S lock is acquired on the partition that is started RO. For a complete description of how the LOCKSIZE clause affects lock attributes, see "What Lock Types DB2 Chooses" on page 5-154.

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.)

Neither the resource multiple nor the reduction of contention is always the same, however. 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.

End of General-use Programming Interface

General-use Programming Interface

LOCKMAX Clause of CREATE and ALTER TABLESPACE

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 5-160. 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.

End of General-use Programming Interface

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

YES can avoid deadlocks but reduces concurrency.

Option to Release Locks for Cursors Defined WITH HOLD

Effect: A subsystem parameter RELCURHL 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: NO

Recommendation: The default, NO, causes DB2 to hold a 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. The default NO is for those cases in which existing applications rely on that particular data lock.

To modify this parameter, edit macro DSN6SPRM, and then run installation job DSNTIJUZ to reassemble the subsystem parameter module.

Option XLOCK for Searched UPDATEs or 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 state 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, there is no need for a lock upgrade request for qualifying rows.

Bind Options

The information under this heading, up to “Specifying Isolation by SQL Statement” on page 5-184, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

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 5-176
- CURRENTDATA, on page 5-179

The ACQUIRE and RELEASE Options

Effects: The ACQUIRE and RELEASE options of bind operations 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 or row 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.

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), with one exception: When you use RELEASE(DEALLOCATE) and KEEPDYNAMIC(YES), and your subsystem is installed with YES for field CACHE DYNAMIC SQL on installation panel DSNTIP4, the RELEASE(DEALLOCATE) option is honored for dynamic SELECT, INSERT, UPDATE and DELETE statements. Locks acquired for dynamic statements are held until one of the following events occurs:

- The application process ends (deallocation).
- The application issues a PREPARE statement with the same statement identifier. (Locks are released at the next commit point.)
- The statement is removed from the cache because it has not been used. (Locks are released at the next commit point.)
- An object that the statement is dependent on is dropped or altered, or a privilege needed by the statement is revoked. (Locks are released at the next commit point.)
- 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). DB2 demotes a gross locks if the reason it was acquired was one of the following:

- The gross lock was acquired because of lock escalation
- The application issued a LOCK TABLE
- The application issued a mass delete (DELETE * FROM 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.

RELEASE(COMMIT)

Releases the lock at the next commit point, unless there are held cursors. 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).

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.

Advantages and Disadvantages of the Combinations

ACQUIRE(ALLOCATE) / RELEASE(DEALLOCATE): Avoids deadlocks by locking all needed resources as soon as the program starts to run.

- 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 for page sets 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 page set 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): 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; except 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): 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 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.
- Locks held by cursors that are defined WITH HOLD are kept past commit points. Except for those, table, partition, or table space locks are released at the next commit point. See “Option to Release Locks for Cursors Defined WITH HOLD” on page 5-171 for more information about an option that lets you avoid the cursor position locks.
- The least restrictive lock needed to execute each SQL statement is used; except 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(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 chooses table and table space locks as nonrestrictive as possible, and 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, the data cannot have changed and no new rows can have been inserted.
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, the data cannot have changed, although additional 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.
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. 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). Rows are returned to an application from either a result table residing in a work file, or directly from the base table. 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.
ISOLATION(UR)	*Uncommitted read*: The application acquires no page or row locks and can run concurrently with most other operations.[10] But the application is in danger of reading data that was changed by another operation but not yet committed. There are restrictions on isolation UR. See Restrictions on page 5-182.

[10] The exceptions are mass delete operations and utility jobs that drain all claim classes.

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 Section 4 of *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 116 shows that all locks are held until the application commits.

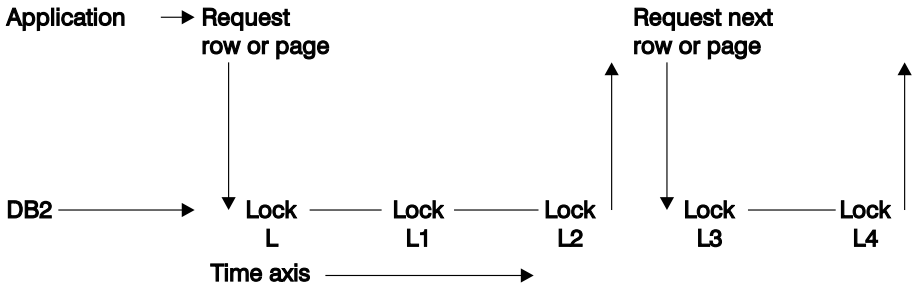


Figure 116. How An Application using RR Isolation Acquires Locks. All locks are held until the application commits.

Applications using 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.

They are not compatible with utility operations that drain all claim classes on a logical partition.

Because so many locks can be taken, lock escalation might take place. Frequent commits releases the locks and can help avoid lock escalation.

With repeatable read, lock promotion occurs for table space scans. DB2 takes the table, partition, or table space lock to avoid accumulating many locks during the scan.

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 5-171.

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 qualify for the stage 1 predicate are locked until the application commits. Figure 117 illustrates this.

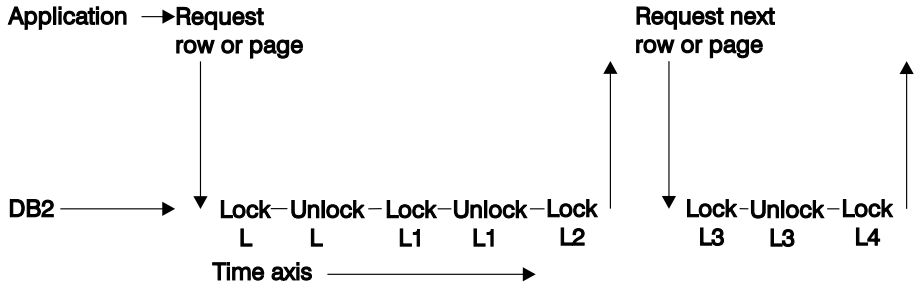


Figure 117. How An Application using RS Isolation Acquires Locks. Locks L2 and L4 are held until the application commits.

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 5-171.

ISOLATION (CS)

Allows maximum concurrency with data integrity. However, after the process leaves a row or page, another process can 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.

CURRENTDATA
This option has two effects:

- For local access, it 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. 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.
- 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.

A cursor is "ambiguous" if DB2 cannot definitely determine whether or not it is read-only.

Problems with ambiguous cursors: If your program has an ambiguous cursor and performs the following operations, your program can 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.

YES
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.

Similarly, if the 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 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 “Using CURRENTDATA(NO) to Ensure Block Fetch” on page 5-320 for information about the effect of CURRENTDATA on block fetch.

Figure 118 shows locking with CURRENTDATA(YES).

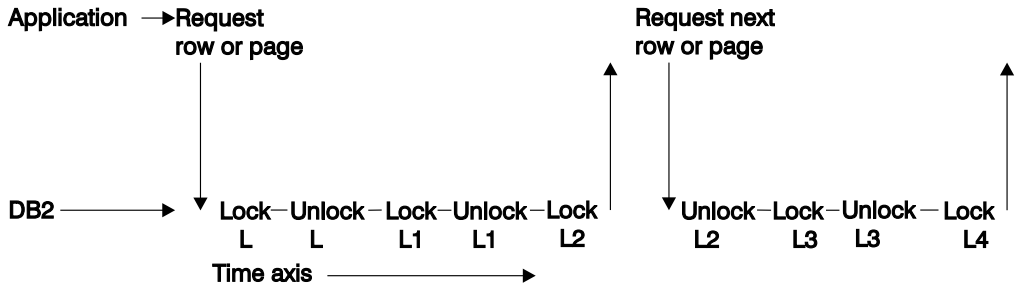


Figure 118. How An Application using Isolation CS with CURRENTDATA(YES) Acquires Locks. This figure shows access to the base table. The L2 and L4 locks are released after DB2 moves to the next row or page. When the application commits, the last lock is released.

NO

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.

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.

To take the best advantage of this method of avoiding locks, make sure all applications that are accessing data concurrently issue COMMITs frequently.

Figure 119 on page 5-181 shows how DB2 can avoid taking locks.

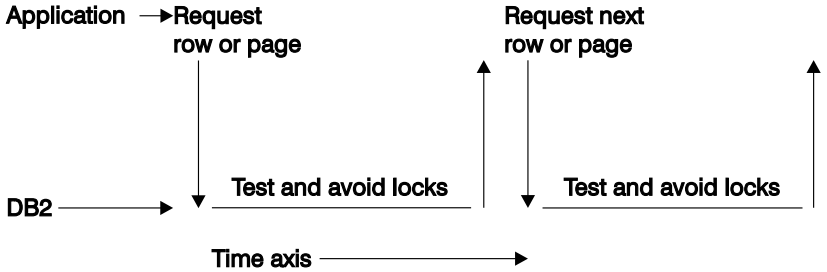


Figure 119. Best Case of Avoiding Locks using CS Isolation with CURRENTDATA(NO). This figure shows access to the base table. If DB2 must take a lock, then locks are released when DB2 moves to the next row or page, or when the application commits (the same as CURRENTDATA(YES)).

For remote access, CURRENTDATA(NO) turns on block fetching for ambiguous cursors.

Table 76 summarizes the effects of CURRENTDATA and cursor type on lock avoidance. See "Using CURRENTDATA(NO) to Ensure Block Fetch" on page 5-320 for information about the effect of CURRENTDATA on block fetch.

Table 76. 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		

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. Now 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

If you bind with isolation UR or specify WITH UR on a statement, DB2 does not choose a type 1 index as a candidate for the access path. For packages bound without specifying an isolation, DB2 might choose a type 1 index for the access path. If it does, and then you run the package under a plan bound with UR, DB2 changes the isolation level to cursor stability.

When Can You Use Uncommitted Read (UR)? Probably in cases like these:

- **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, they allow updates to 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 for them to restrict their queries to times when no updates are being made, and then they can run with UR isolation.

- **When an error is acceptable**.

  Example: Spiffy wants to do some statistical analysis on their employee data. A typical question is, “What is the average salary by sex within education level?” They estimate that reading an occasional uncommitted record cannot affect the averages much, so they use UR isolation.

- **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 3-128.

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 two types of lock:

- 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 so 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.

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 77 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 77. 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 subsystem parameter RELCURHL, described in “Option to Release Locks for Cursors Defined WITH HOLD” on page 5-171, the page or row lock, if that lock is not successfully avoided through lock avoidance, is held past the commit point. 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 according to the isolation level at which it was acquired: for CS, when all cursors on the page are moved or closed; for RR or RS, at the next commit point, provided that no cursor is still positioned on that page or row.

| If you specify YES for RELCURHL, 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.

Specifying Isolation by SQL Statement

The information under this heading, up to “The Statement LOCK TABLE” on page 5-185 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

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 DSN8510.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 subselect
  - 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 “Controlling Concurrency for Utilities and Commands” on page 5-186 is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

Purpose: The statement overrides DB2 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.

When To Use It: 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.

Example: 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

The change affects about 15% of the employees, so 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 on Table Spaces of Different Types on page 5-146 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.

Effects: Table 78 shows the modes of locks acquired in segmented and nonsegmented table spaces for the SHARE and EXCLUSIVE modes of LOCK TABLE. LOCK TABLE has no effect on locks acquired at a remote server.

Table 78. Modes of Locks Acquired by LOCK TABLE. Partition locks behave the same as locks on a nonsegmented table space.

		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.

Duration of the Locks: The bind option RELEASE determines when locks acquired by LOCK TABLE or LOCK TABLE with the PART option are released.

Option	**Releases locks ...**
RELEASE(COMMIT)	At the next commit point. Page or row locking resumes in the next unit of work.
RELEASE(DEALLOCATE)	Only when the program ends.

Controlling Concurrency for Utilities and Commands

DB2 utilities and commands can take over access to some objects independently of any transaction locks that are held on the object.

Objects Subject to Takeover

- Simple and segmented table spaces
- Partitions of table spaces
- Nonpartitioned index spaces
- Partitions of index spaces
- Logical partitions of nonpartitioned type 2 indexes

This section describes the effects of those takeovers under:

- “Definition of Claims and Drains” on page 5-187
- “Usage of Drain Locks” on page 5-188
- “Utility Locks on the Catalog and Directory” on page 5-188

- "Compatibility of Utilities" on page 5-189
- "Utility Operations with Nonpartitioned Indexes" on page 5-190

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.

(There is an exception, however; 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 5-184.)

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 utility controls access to the partition. The applications that were drained can still hold transaction locks on the partition, but they cannot make new claims until the utility has finished.

Claim Classes Drained

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.

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. In order 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 when all held cursors are released.

A drain lock also prevents new claimers from accessing an object while a drainer has control of it.

Types of Drain Locks

There are three types of drain locks on an object, which 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.[11]

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

---

[11] 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.

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.

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. (An exception occurs when the utility processes a nonpartitioned type 1 index; for details, see “Utility Operations with Nonpartitioned Indexes” on page 5-190.)

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.

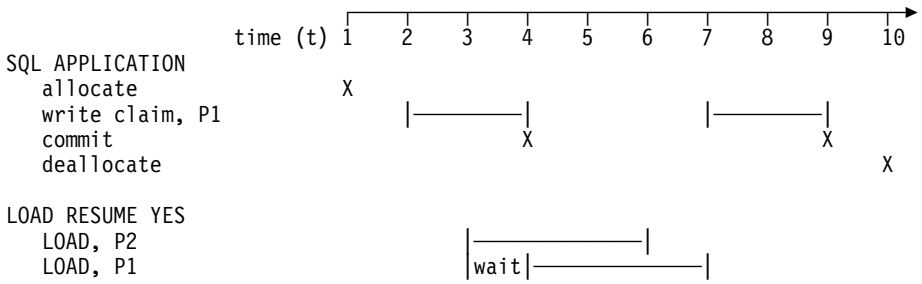
  An exception is a case in which one utility must update a catalog or directory table space that is not the direct target of the utility. For example, the LOAD utility on a user table space updates DSNDB06.SYSCOPY. Therefore, other utilities that that have DSNDB06.SYSCOPY as a target might not be compatible.

- 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 *Utility Guide and Reference*.

Figure 120 on page 5-190 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 120. SQL and Utility Concurrency. Two LOAD jobs execute concurrently on two partitions of a table space

Controlling concurrency during REORG

Utility Operations with Nonpartitioned Indexes

In the example of Figure 120, 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 nonpartitioned index, an entry can refer to any partition in the underlying table space. That fact presents no problem if the index is a type 2 index. Using type 2 indexes, DB2 can process a set of entries of a nonpartitioned index that all refer to a single partition and achieve the same results as for a partition of a partitioned index. (Such a set of entries is called a *logical partition* of the index.)

If the example included a nonpartitioned type 1 index, however, the two LOAD jobs could not build it concurrently. Each needs a drain lock on the entire index; the first job to reach that phase acquires the lock and locks out the other job.

Monitoring DB2 Locking

If you have problems with suspensions, timeouts, or deadlocks, you will want to monitor DB2's use of locks.

Use the -DISPLAY DATABASE command 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 type 2 indexes. For an example, see "Monitoring Databases" on page 4-25.

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 5-192.

The final section of this chapter gives an example of resolving a particular locking problem. See "Concurrency Scenario" on page 5-193.

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 5-192, is Product-sensitive Programming Interface and Associated Guidance Information, as defined in "Notices" on page xi.

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 Information from EXPLAIN" on page 5-262.
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 79 on page 5-192, 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 5-10. Using EXPLAIN to Improve SQL Performance" on page 5-261.

Table 79. 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.					
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 121 on page 5-193 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 or row 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 (LATCH ONLY)            0     0.00     0.00     0.00  ||   ESCAL.(SHAR)        0
SUSPENSIONS (OTHER)                 0     0.00     0.00     0.00  ||   ESCAL.(EXCL)        0
                                                                  ||   MAX.LCK HELD        2
TIMEOUTS                            0     0.00     0.00     0.00  ||   LOCK REQUEST        8
DEADLOCKS                           1     0.64     0.50     0.20  ||   UNLOCK REQST        2
                                                                  ||   QUERY REQST         0
LOCK REQUESTS                      17    10.92     8.50     3.40  ||   CHANGE REQST        5
UNLOCK REQUESTS                    12     7.71     6.00     2.40  ||   OTHER REQST         0
QUERY REQUESTS                      0     0.00     0.00     0.00  ||   LOCK SUSP.          1
CHANGE REQUESTS                     5     3.21     2.50     1.00  ||   LATCH SUSP.         0
OTHER REQUESTS                      0     0.00     0.00     0.00  ||   OTHER SUSP.         0
                                                                  ||   TOTAL SUSP.         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 121. Locking Activity Blocks from Statistics Trace and Accounting Trace

Concurrency Scenario

The concurrency problem analyzed in this section illustrates the approach described in “A General Approach to Problem Analysis in DB2” on page 5-32. It follows the CONCURRENCY PROBLEM branch of Figure 96 on page 5-35 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

First, we describe the scenario. Next, we show how each report helps to analyze the situation. We end with some general information about corrective approaches.

Scenario Description

An application (PARALLEL), 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. This allows 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 122 shows a portion of Accounting Report - Long.

```
   LOCATION: SYS1DSN2                    DB2 PERFORMANCE MONITOR (V4 R2)                    PAGE: 1-1
      GROUP: DSN2                           ACCOUNTING REPORT - LONG                REQUESTED FROM: NOT SPECIFIED
     MEMBER: SE11                                                                          TO: NOT SPECIFIED
  SUBSYSTEM: SE11                                ORDER: PLANNAME                  INTERVAL FROM: 08/14/96 11:41:10.15
DB2 VERSION: V4 R2                               SCOPE: MEMBER                           TO: 08/14/96 11:41:10.15

PLANNAME: PARALLEL

AVERAGE       APPL (CLASS 1)   DB2 (CLASS 2)   IFI (CLASS 5)    CLASS 3 SUSP.   AVERAGE TIME  AV.EVENT    HIGHLIGHTS  D
------------  --------------  --------------  --------------    --------------  ------------  --------    --------------------------
                  A                B                                                  C
ELAPSED TIME     5:03.575400     5:03.383300            N/P    LOCK/LATCH       5:03.277805      1.00    #OCCURRENCES    :        2
CPU TIME            0.046199        0.021565            N/P    SYNCHRON. I/O       0.000000      0.00    #ALLIEDS        :        2
 TCB                0.046199        0.021565            N/P    OTHER READ I/O      0.000000      0.00    #ALLIEDS DISTRIB:        0
 TCB-STPROC         0.000000        0.000000            N/A    OTHER WRTE I/O      0.000000      0.00    #DBATS          :        0
 PAR.TASKS          0.000000        0.000000            N/A    SER.TASK SWTCH      0.082205      5.00    #DBATS DISTRIB. :        0
SUSPEND TIME             N/A     5:03.360010            N/A    ARC.LOG(QUIES)      0.000000      0.00    #NO PROGRAM DATA:        0
 TCB                     N/A     5:03.360010            N/A    ARC.LOG READ        0.000000      0.00    #NORMAL TERMINAT:        2
 PAR.TASKS               N/A        0.000000            N/A    DRAIN LOCK          0.000000      0.00    #ABNORMAL TERMIN:        0
NOT ACCOUNT.             N/A        0.001725            N/A    CLAIM RELEASE       0.000000      0.00    #CP/X PARALLEL. :        0
DB2 ENT/EXIT             N/A            5.00            N/A    PAGE LATCH          0.000000      0.00    #IO PARALLELISM :        0
EN/EX-STPROC             N/A            0.00            N/A    STORED PROC.        0.000000      0.00    #INCREMENT. BIND:        0
DCAPT.DESCR.             N/A             N/A            N/P    NOTIFY MSGS.        0.000000      0.00    #COMMITS        :        2
LOG EXTRACT.             N/A             N/A            N/P    GLOBAL CONT.        0.000000      0.00    #ROLLBACKS      :        1
                                                               TOTAL CLASS 3    5:03.360010      6.00    UPDATE/COMMIT   :     0.00

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    INDEX            0       0       0    ESCAL.(SHARED)      0.00         0
DELETE        0.00         0    SET CURR.SQLID        0    TABLESPACE       0       0       0    ESCAL.(EXCLUS)      0.00         0
                                SET HOST VAR.         0    DATABASE         0       0       0    MAX LOCKS HELD      1.00         1
DESCRIBE      0.00         0    SET CUR.DEGREE        0    STOGROUP         0       0       0    LOCK REQUEST        5.00        10
DESC.TBL      0.00         0    SET RULES             0    SYNONYM          0       0     N/A    UNLOCK REQUEST      5.00        10
PREPARE       0.00         0    CONNECT TYPE 1        0    VIEW             0       0     N/A    QUERY REQUEST       0.00         0
OPEN          1.00         2    CONNECT TYPE 2        0    ALIAS            0       0     N/A    CHANGE REQUEST      1.00         2
FETCH         0.00         0    SET CONNECTION        0    PACKAGE        N/A       0     N/A    OTHER REQUEST       0.00         0
CLOSE         0.00         0    RELEASE               0                                          LOCK SUSPENS.       1.00         2
                                CALL                  0    TOTAL            0       0       0    LATCH SUSPENS.      0.00         0
                                ASSOC LOCATORS        0
DML-ALL       1.00         2    ALLOC CURSOR          0    COMMENT ON       0                    OTHER SUSPENS.      0.00         0
                                DCL-ALL               0    LABEL ON         0                    TOTAL SUSPENS.      1.00         2
```

Figure 122. Excerpt from Accounting Report - Long

Accounting Report - Long shows the average elapsed times and the average number of suspensions per plan execution. In Figure 122:

- 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 123 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 123. 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.
- Average elapsed time (AET) per suspension.

The report also shows the reason for the suspensions:

Reason	Includes...
LOCAL	Contention for a local resource.
LATCH	Contention for latches within IRLM; suspension is usually brief.
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 124 on page 5-196 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  CORRNAME CORRNMBR  HOLDER WAITER
----------------- --------- ----------------------- -------- ---------  -------- -------- -------- -------- --------  ------ ------
FPB
 PARALLEL         PARTITION DB  =PARADABA                 2         0  N/P      DSNESPRR TSO      EOA      'BLANK'        2      0
                            OB  =TAB1TS
                            PART= 1
                  ** LOCKOUTS FOR PARALLEL   **           2         0
```

Figure 124. Portion of DB2 PM Locking Report - Lockout

Lockout Trace

Figure 125 shows the DB2 PM Locking Trace - Lockout report.

This report produces one line for each lock contender. It shows the contenders, 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   CORRNAME=EOA
                                                                                          DURATION=COMMIT     CORRNMBR='BLANK'
                                                                                          STATE   =X

FPB      FPBPARAL TSO          15:30:32.97267562 TIMEOUT  PARTITION DB  =PARADABA           REQUEST =LOCK       UNCONDITIONAL
FPB      'BLANK'  AB09C65528E6 N/P                                   OB  =TAB1TS           STATE   =IS         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   CORRNAME=EOA
                                                                                          DURATION=COMMIT     CORRNMBR='BLANK'
                                                                                          STATE   =X

LOCKING TRACE COMPLETE
```

Figure 125. 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.

Making 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 123 on page 5-195, the number of suspensions is low (only 2) and both have ended in a timeout. Rather than use the DB2 performance trace for locking, the preferred option is to use 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

Here we examine two different deadlock scenarios and tell how to use 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, we see contention for locks 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. There are two transactions and two resources 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:

```
                  |  LOC2A - obtains a U lock on page 2 in table DEPT,
                  |          to open its cursor for update
                  |
                  |  LOC2B - obtains a U lock on a page 8 in table PROJ,
                  |          to open its cursor for update
             Time |
                  |  LOC2A - attempts to access page 8, to open its cursor
                  |          (Cannot proceed, because of the lock held by LOC2B.)
                  |
                  |  LOC2B - attempts to access page 2, to open its cursor
                  |          (Cannot proceed, because of the lock held by LOC2B.)
                  V
```

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 126 shows the DB2 PM Locking Trace - Deadlock report produced for this situation.

From the report we see 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      CORRNAME=RUNLOC2A
                                                                                  DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                  STATE   =U
                                                                                  ---------------- WAITER ----------------
                                                                                  LUW='BLANK'.EGTVLU2.AA65FEDC1022
                                                                                  MEMBER  =DB1A       CONNECT =BATCH
                                                                                  PLANNAME=LOC2B      CORRNAME=RUNLOC2B
                                                                                  DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                  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      CORRNAME=RUNLOC2B
                                                                                    DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                    STATE   =U
                                                                                    ---------------- WAITER -------*VICTIM*-
                                                                                    LUW='BLANK'.EGTVLU2.AADD32FD8A8C
                                                                                    MEMBER  =DB1A       CONNECT =BATCH
                                                                                    PLANNAME=LOC2A      CORRNAME=RUNLOC2A
                                                                                    DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                    REQUEST =LOCK       WORTH   =   17
                                                                                    STATE   =U
```

Figure 126. 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, we note that the entry in the trace, identified at **A**, is LOC2A. That is the selected thread whose work is rolled back to let the other proceed (the "victim").

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. There are three transactions and three resources 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.

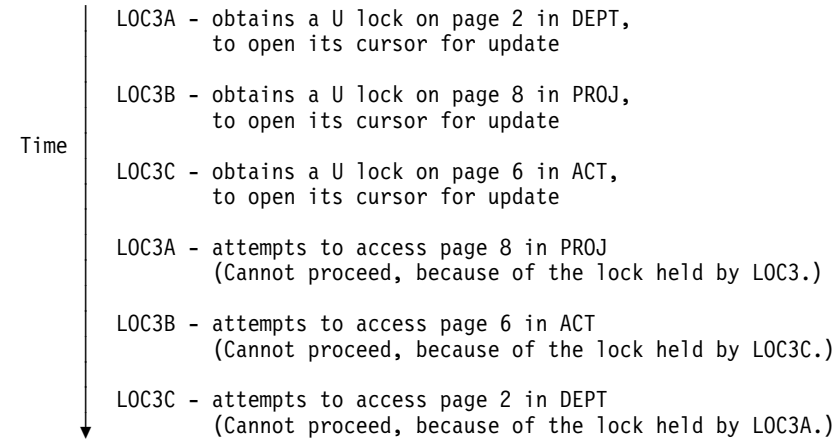
LOC3B

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:



DB2 rolls back one of the transactions, plan LOC3C, and releases its locks. That allows another transaction to complete and release its locks. And that allows the third (and final) transaction to complete.

Figure 127 on page 5-201 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      CORRNAME=RUNLOC3B
                                                                                           DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                           STATE   =U
                                                                                           ---------- WAITER ----------
                                                                                           LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC3A      CORRNAME=RUNLOC3A
                                                                                           DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                           REQUEST =LOCK       WORTH   =   18
                                                                                           STATE   =U

                                                   DATAPAGE  DB  =DSN8D42A                 HASH    =X'01060711'
                                                           OB  =ACT                        ---------- BLOCKER IS HOLDER -----
                                                           PAGE=X'000006'                  LUW='BLANK'.EGTVLU2.AADE2CF16F34
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC3C      CORRNAME=RUNLOC3C
                                                                                           DURATION=COMMIT     CORRNMBR='BLANK'
                                                                                           STATE   =X
                                                                                           ---------- WAITER ----------
                                                                                           LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC3B      CORRNAME=RUNLOC3B
                                                                                           DURATION=COMMIT     CORRNMBR='BLANK'
                                                                                           REQUEST =LOCK       WORTH   =   18
                                                                                           STATE   =X
⋮
SYSADM   RUNLOC3C TSO                              DATAPAGE  DB  =DSN8D42A                 HASH    =X'01060304'
SYSADM   'BLANK'  AADE2CF16F34                             OB  =DEPT                       ---------- BLOCKER IS HOLDER------
LOC3C    BATCH                                             PAGE=X'000002'                LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC3A      CORRNAME=RUNLOC3A
                                                                                           DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                           STATE   =U
                                                                                           ---------- WAITER -------*VICTIM*-
                                                                                           LUW='BLANK'.EGTVLU2.AADE2CF16F34
                                                                                           MEMBER  =DB1A       CONNECT =BATCH
                                                                                           PLANNAME=LOC3C      CORRNAME=RUNLOC3C
                                                                                           DURATION=MANUAL     CORRNMBR='BLANK'
                                                                                           REQUEST =LOCK       WORTH   =   17
                                                                                           STATE   =U
```

Figure 127. Deadlock scenario 2: Three transactions and three resources

Chapter 5-8. 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 xi.

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 5-206
- “Using Host Variables Efficiently” on page 5-224
- “Writing Efficient Subqueries” on page 5-228

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 5-233 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 80 on page 5-211 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 5-228 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 we will refer to as P1) is evaluated 1,000 times, and the predicate in subquery 2 (which we will refer 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 5-228 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 5-219 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?” on page 5-275 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.

If your query involves the functions MAX or MIN, refer to “One-Fetch Access (ACCESSTYPE=I1)” on page 5-281 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 5-224 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 5-220 for ideas on what to do about it.

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

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 5-207
- Whether it is **indexable**, as described in "Indexable and Nonindexable Predicates" on page 5-207
- Whether it is **stage 1** or **stage 2**

There are special considerations for "Predicates in the ON Clause" on page 5-209.

Definitions: We identify predicates 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" on page 5-282.

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 DSN8510.EMP WHERE LASTNAME = 'SMITH';
```

The following predicate cannot be a matching predicate, because it is not indexable.

```
SELECT * FROM DSN8510.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 80 on page 5-211 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.

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 range predicates also.

For left and right outer joins, and for inner joins, 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, most predicates in the WHERE clause are evaluated AFTER the join, and are therefore 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 DSN8510.EMP
   WHERE EDLEVEL  > 100) AS X FULL JOIN DSN8510.DEPT
      ON X.WORKDEPT  = DSN8510.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” on page 5-282.

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 5-215.

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.
2. Other stage 1 predicates are applied next.
   a. 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*. In general, DB2 chooses the most restrictive predicate as the matching predicate. All other predicates become index screening predicates.
   b. 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 80 on page 5-211 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 80. Predicate Types and Processing

Predicate Type	Index-able?	Stage 1?	Notes
COL = *value*	Y	Y	
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	
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	14
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

Table 80. Predicate Types and Processing

Predicate Type	Index-able?	Stage 1?	Notes
COL=(*non subq*)	Y	Y	13
COL = ANY (*non subq*)	N	N	
COL = ALL (*non subq*)	N	N	
COL *op* (*non subq*)	Y	Y	13
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	
COL 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	
COL NOT IN (*cor subq*)	N	N	
EXISTS (*subq*)	N	N	
NOT EXISTS (*subq*)	N	N	
COL = *expression*	N	N	7
expression = *value*	N	N	
expression <> *value*	N	N	
expression op value	N	N	
expression op (subquery)	N	N	

Notes to Table 80:

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 column lengths, such as CHAR(5) and CHAR(20)
   - Different precisions, such as DECIMAL(7,3) and DECIMAL(7,4).

   The following are considered to be columns of the same type:
   - 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. Not indexable and not stage 1 if COL is not null and the noncorrelated
subquery SELECT clause entry can be null.

14. 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 80 on page 5-211. 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 not indexable except by a union of RID lists from two indexes and cannot be considered for matching index access.

- 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.

- WHERE C1>5 OR C2=7

  Both predicates are stage 1 but not Boolean terms. The compound is not indexable except by a union of RID lists from two indexes and cannot be considered for matching index access.

- 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 not indexable but stage 1; it is evaluated at 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 T3.COL2=T4.COL2

  Assume that T1.COL1 and T2.COL1 have the same data types, and T3.COL2 and T4.COL2 have the same data types. If T1.COL1 and T2.COL1 have different nullability attributes, but T3.COL2 and T4.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 T3.COL2 and T4.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 “Using RUNSTATS to Monitor and Update Statistics” on page 5-249. For information on updating the catalog manually, see “Updating Catalog Statistics” on page 5-241.

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” on page 5-216
- “Interpolation Formulas” on page 5-216
- “Filter Factors for All Distributions” on page 5-218

Default Filter Factors for Simple Predicates

Table 81 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 81 (Page 1 of 2). DB2 Default Filter Factors by Predicate Type

Predicate Type	Filter Factor
Col = literal	1/25
Col IS NULL	1/25

Table 81 (Page 2 of 2). DB2 Default Filter Factors by Predicate Type

Predicate Type	Filter Factor
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 82 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 82. 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
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 81 on page 5-215 and Table 82 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, host variables, or special registers.
  - COLCARDF < 4.

Table 83 on page 5-218 shows interpolation defaults for the operators <, <=, >, >= and for LIKE and BETWEEN.

Table 83. 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 84 lists the types of predicates on which these statistics are used.

Table 84. Predicates for Which Distribution Statistics are Used

Type of Statistic	Single Column or Concatenated Columns	Predicates
Frequency	Single	COL=*literal* COL IS NULL COL IN (*literal-list*) COL *op literal* COL BETWEEN *literal* AND *literal*
Frequency	Concatenated	COL=*literal*
Cardinality	Single	COL=*literal* COL IS NULL COL IN (*literal-list*) COL *op literal* COL BETWEEN *literal* AND *literal* COL=*host-variable* COL1=COL2
Cardinality	Concatenated	COL=*literal* COL=*:host-variable* 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 Section 2 of *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 80 on page 5-211 is also helpful.

Predicate Modifications

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)`.

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: DB2 generates predicates for transitive closure if:

- The query has an equal type predicate: C1=C2. This could be a join predicate or a local predicate.
- The query has another equal or range type predicate on one of the columns in the first predicate: C1 BETWEEN 3 AND 5. This predicate cannot be a LIKE predicate and must be a Boolean term predicate.

When these conditions are met, DB2 generates a new predicate, whether or not it already exists in the WHERE clause. In the above case, DB2 generates the predicate C2 BETWEEN 3 AND 5.

Extra join predicates are not generated if more than nine tables are joined in a query.

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 85. Effects of Column Correlation on Matching Columns

	INDEX 1	**INDEX 2**
Matching Predicates	Predicate1 CITY=FRESNO AND STATE=CA	Predicate2 DEPTNO=A345 AND SEX=F
Matching Columns	2	2
DB2 estimate for matching columns (Filter Factor)	column=CITY, COLCARDF=4 Filter Factor=1/4 column=STATE, COLCARDF=3 Filter Factor=1/3	column=DEPTNO, COLCARDF=4 Filter Factor=1/4 column=SEX, COLCARDF=2 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 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 BETTER INDEX CHOICE (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 5-279 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 5-221) with the following indexes on table CREWINFO (page 5-220):

```
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 5-221) 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 5-233.

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 Data Access Statistics Current” on page 5-37 for information on using RUNSTATS to influence access path selection. See “Updating

Catalog Statistics” on page 5-241 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 5-215.

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, STMTNO, SEQNO, TEXT
  FROM   SYSIBM.SYSSTMT
  WHERE STATUS IN ('B','F','G','J')
  ORDER BY PLNAME, STMTNO, SEQNO;

SELECT COLLID, NAME, VERSION, STMTNO, SEQNO, STMT
  FROM SYSIBM.SYSPACKSTMT
  WHERE STATUS IN ('B','F','G','J')
  ORDER BY COLLID, NAME, VERSION, STMTNO, 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 5-224 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 DSN8510.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 DSN8510.EMP
       WHERE SEX = 'M';
    END;
  WHEN ('F')
    DO;
      EXEC SQL SELECT * FROM DSN8510.EMP
       WHERE SEX = 'F';
    END;
  OTHERWISE
    DO:
      EXEC SQL SELECT * FROM DSN8510.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" on page 5-229
- "Noncorrelated Subqueries" on page 5-230

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 5-231

Finally, for a comparison of the three methods as applied to a single task, see:

- "Subquery Tuning" on page 5-232

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 DSN8510.EMP X
   WHERE  JOB = 'DESIGNER'
      AND  EXISTS (SELECT 1
                   FROM   DSN8510.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:

< <= > >= = <> 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 DSN8510.EMP
  WHERE  JOB = 'DESIGNER'
    AND  WORKDEPT IN (SELECT DEPTNO
                      FROM   DSN8510.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:

< <= > >= = <> EXISTS NOT EXISTS

The following noncorrelated subquery returns a single value:

```
SELECT *
FROM   DSN8510.EMP
WHERE  JOB = 'DESIGNER'
  AND  WORKDEPT <= (SELECT MAX(DEPTNO)
                    FROM   DSN8510.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 DSN8510.EMP
   WHERE  JOB = 'DESIGNER'
     AND  WORKDEPT <= ANY (SELECT DEPTNO
                           FROM   DSN8510.PROJ
                           WHERE  MAJPROJ = 'MA2100');
```

DB2 calculates the maximum value for DEPTNO from table DSN8510.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?” on page 5-275.

Subquery Transformation into Join

A subquery can be transformed into a join between the result table of the subquery and the result table of the outer query, provided that the transformation does not introduce redundancy.

DB2 makes that transformation only if:

- 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 subquery select list has only one column, guaranteed by a unique index to have unique values.

- The transformation results in 15 or fewer tables in the join.
- 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.)

Example: The following subquery could be transformed into a join:

```
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';
```

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? (QBLOCKNO Value)” on page 5-274.

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 DSN8510.EMP.* FROM DSN8510.EMP, DSN8510.PROJ
   WHERE  JOB = 'DESIGNER'
     AND  WORKDEPT = DEPTNO
     AND  MAJPROJ = 'MA2100';
```

Query B: A correlated subquery

```
SELECT * FROM DSN8510.EMP X
   WHERE  JOB = 'DESIGNER'
     AND  EXISTS (SELECT 1 FROM DSN8510.PROJ
                  WHERE  DEPTNO = X.WORKDEPT
                    AND  MAJPROJ = 'MA2100');
```

Query C: A noncorrelated subquery

```
SELECT * FROM DSN8510.EMP
    WHERE  JOB = 'DESIGNER'
      AND  WORKDEPT IN (SELECT DEPTNO FROM DSN8510.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. 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.

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
- “Using OPTIMIZE FOR n ROWS” on page 5-234
- “Reducing the Number of Matching Columns” on page 5-236
- “Adding Extra Local Predicates” on page 5-239
- “Changing an Inner Join into an Outer Join” on page 5-240

- "Rearranging the Order of Tables in a FROM Clause" on page 5-240
- "Updating Catalog Statistics" on page 5-241

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 Information from EXPLAIN" on page 5-262.

Using 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 Section 4 of *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 Section 4 of *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
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 Section 4 of *Call Level Interface 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 EMPLOYEE
  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 EMPLOYEE
  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 both local and distributed queries, specify OPTIMIZE FOR *n* ROWS only in applications that frequently read 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
- Limiting the number of rows that flow across the network on any given transmission

To influence the access path most, specify OPTIMIZE for 1 ROW. This value does not have a detrimental effect on distributed queries. To increase the number of rows returned in a single network transmission, you can specify a larger value for *n*, such as the number of rows that fit on a terminal screen, without negatively influencing the access path.

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. Reducing the number of best matching columns for a statement has varying effects depending on how the predicate is altered.

Changing an Equal Predicate to a BETWEEN Predicate: Use the original query on table CREWINFO:

Old Query 1

```
SELECT ... FROM CREWINFO WHERE
  CITY = 'FRESNO' AND STATE = 'CA'
  AND DEPTNO = 'A345' AND SEX = 'F';
```

with Index 1 (CITY,STATE) and Index 2 (DEPTNO,SEX) but with the following change:

New Query 1

```
SELECT ... FROM CREWINFO WHERE
  CITY BETWEEN 'FRESNO' AND 'FRESNO'    (MODIFIED PREDICATE)
  AND STATE = 'CA'
  AND DEPTNO = 'A345' AND SEX = 'F';    (PREDICATE2)
```

The original Query 1 had a MATCHCOLS value of 2 because there were matching predicates on the two leading columns of the index. The new Query 1 has a MATCHCOLS value of 1 because of the BETWEEN predicate on the leading index column of Index 1. Index 2, which still has MATCHCOLS of 2, is now the optimal choice.

DB2 might not choose Index 2 if there are statistics for table CREWINFO. If statistics exist, the choice of index depends on the filter factors of these two predicates:

```
CITY BETWEEN 'FRESNO' AND 'FRESNO'
DEPTNO = 'A345' AND SEX = 'F'
```

Discouraging Use of a Particular Index: You can discourage a particular index from being used by reducing the number of MATCHCOLS it has. Consider the example in Figure 128 on page 5-239, 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 checked as a stage 2 predicate, which is more expensive than a stage 1 predicate. However, if most of the filtering is already done by the first and second predicates, having the third predicate as a stage 2 predicate should not degrade performance significantly.

This technique for discouraging index usage can be used in the same way to discourage the use of multiple index access. Changing a join predicate into a stage 2 predicate would prevent it from being used during a join.

There are many ways to make a predicate stage 2. The recommended way is to make the predicate a non-Boolean term by adding an OR predicate as follows:

Stage 1	Stage 2
`T1.C1=T2.C2`	`(T1.C1=T2.C2 OR 0=1)`
`T1.C1=5`	`(T1.C1=5 OR 0=1)`

Adding this OR predicate does not affect the result of the query. It is valid for use with columns of all data types, and causes 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

columnn	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			
index	matchcols	filter factor	data rows	index	matchcols	filter factor	data rows
ix2	2	.02*.02	40	ix2	2	.02*.50	1000
ix1	1	.001	100	ix1	1	.001	100

Figure 128. 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.

Changing an Inner Join into an Outer Join

You can discourage the use of hybrid joins by making your inner join statement into an outer join statement, then using a WHERE clause to eliminate the unneeded rows. An outer join does not use the hybrid join method. You can also make use of outer join to force a particular join sequence.

For example, suppose you want to obtain the results of the following inner join operation on the PARTS and PRODUCTS tables:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS, PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

DB2 creates a more efficient access path if the PARTS table is the outer table in the join operation. To make the PARTS table the outer table, use a left outer join operation. Include a WHERE clause in the query to remove the extraneous rows:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS LEFT OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
    WHERE PRODUCTS.PROD# IS NOT NULL;
```

For more information on outer joins, see Section 2 of *Application Programming and SQL Guide*.

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 5-10. Using EXPLAIN to Improve SQL Performance” on page 5-261. 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 86 on page 5-244.

The example shown in Figure 128 on page 5-239, 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 128 on page 5-239, 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 "Using RUNSTATS to Monitor and Update Statistics" on page 5-249 and Section 2 of *Utility Guide and Reference*for more information.
- Update the catalog statistics manually.

A good catalog table to 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 *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);
```

Please remember that 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 System Parameter to Enhance Outer Join Performance

DB2 provides a system parameter that enables several performance enhancements for outer join operations. Those enhancements include:

- Optimization of view merging and table expression merging
- Increasing the number of predicates that can be evaluated before join operations
- Optimization of transitive closure across join operations

The system parameter is named OJPERFEH and is in macro DSN6SPRM. To enable the performance enhancements, add OJPERFEH=YES to the DSN6SPRM invocation in step DSNTIZA of installation job DSNTIJUZ. Then rerun DSNTIJUZ and restart DB2.

Chapter 5-9. Maintaining Statistics in the Catalog

Statistics stored in the DB2 catalog help DB2 determine the access paths selected for your SQL statements.

- Table 86 on page 5-244 lists the tables and columns in the catalog that contain those statistics.
- "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 all of the values.

- "Statistics for Created Temporary Tables" on page 5-249 explains how to set default statistical values for created temporary tables.
- "Using RUNSTATS to Monitor and Update Statistics" on page 5-249 has some advice about running RUNSTATS.
- "Updating the Catalog" on page 5-250 warns of items to watch for if you make your own updates.
- "Querying the Catalog for Statistics" on page 5-252 tells how to find out what values are in place.

The statistics have other uses as well as access path selection. For those, see

- "Improving Index and Table Space Access" on page 5-253, and
- "Modeling Your Production System" on page 5-258.

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 *Data Sharing: Planning and Administration* .

Statistics Used for Access Path Selection

Table 86 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 86 (Page 1 of 4). Catalog Data Used for Access Path Selection or Collected by RUNSTATS. Some Version 4 columns are no longer used in Version 5 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?	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				
COLVALUE	Yes	Yes	Yes	Frequently occurring value in a non-uniform 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. .15 indicates that 15% of the rows contain the value.
TYPE	Yes	Yes	Yes	The type of statistics gathered, either cardinality (C) or frequent value (F).
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.
NUMCOLUMNS	Yes	Yes	Yes	The number of columns associated with the statistics. The **default value** is 1.
SYSIBM.SYSCOLDISTSTATS: contains statistics by partition.				
COLVALUE	Yes	Yes	No	Frequently occurring value in a non-uniform 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. .15 indicates that 15% of the rows contain the value.
TYPE	Yes	Yes	No	The type of statistics gathered, either cardinality (C) or frequent value (F).
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.
NUMCOLUMNS	Yes	Yes	No	The number of columns associated with the statistics. The **default value** is 1.
SYSIBM.SYSCOLSTATS: contains statistics by partition.				

Table 86 (Page 2 of 4). Catalog Data Used for Access Path Selection or Collected by RUNSTATS. Some Version 4 columns are no longer used in Version 5 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?	Description
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.
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
HIGH2KEY	Yes	Yes	No	First 8 bytes of the second highest value of the column within the partition
LOWKEY	Yes	Yes	No	First 8 bytes of the lowest value of the column within the partition
LOW2KEY	Yes	Yes	No	First 8 bytes of the second lowest value of the column within the partition
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)
HIGH2KEY	Yes	Yes	Yes	First 8 bytes of the second highest value in this column
LOW2KEY	Yes	Yes	Yes	First 8 bytes of the second lowest value in this column
SYSIBM.SYSINDEXES				
CLUSTERED	Yes	No	No	Whether the table is actually clustered by the index
CLUSTERING	No	No	Yes	Whether the index was created using CLUSTER
CLUSTERRATIO	Yes	Yes	Yes	Percentage of rows that are in clustering order
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)
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)

SYSIBM.SYSINDEXPART: contains statistics for space utilization.

Table 86 (Page 3 of 4). Catalog Data Used for Access Path Selection or Collected by RUNSTATS. Some Version 4 columns are no longer used in Version 5 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?	Description
CARDF	Yes	No	No	Number of rows referenced by the index or partition
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. See "How LEAFDIST is Calculated" on page 5-256 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 optimial position because of an insert into a full page
PQTY	Yes	No	No	The primary space allocation in 4K blocks for the dataset
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 pageset contains multiple pieces)
SQTY	Yes	No	No	The secondary space allocation in 4K blocks for the dataset
SYSIBM.SYSINDEXSTATS: contains statistics by partition.				
CLUSTERRATIO	Yes	Yes	No	Percentage of rows that are in clustering order
FIRSTKEYCARD	Yes	Yes	No	Number of distinct values of the first key column, or an esimate if updated while collecting statistics on a single partition
FULLKEYCARD	Yes	Yes	No	Number of distinct values of the full key
NLEAF	Yes	Yes	No	Number of leaf pages in the index
NLEVELS	Yes	Yes	No	Number of levels in the index tree
KEYCOUNT	Yes	Yes	No	Number of rows in the partition
SYSIBM.SYSTABLEPART: contains statistics for space utilization.				
CARD	Yes	No	No	Total number of rows in the table space or partition
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

| *Table 86 (Page 4 of 4). Catalog Data Used for Access Path Selection or Collected by RUNSTATS. Some Version 4 columns are no longer used in Version 5 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?	Description
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 dataset
# 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 pageset contains multiple pieces)
# SQTY #	Yes	No	No	The secondary space allocation in 4K blocks for the dataset
SYSIBM.SYSTABLES:				
CARDF	Yes	Yes	Yes	Total number of rows in the 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))
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
SYSIBM.SYSTABLESPACE:				
NACTIVE	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 **default value** (CEILING(1 + CARD/20))
SYSIBM.SYSTABSTATS: contains statistics by partition				
CARD	Yes	Yes	Yes	Total number of rows in the partition; -1 to trigger DB2's use of the **default value**(10 000)
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
NACTIVE	Yes	Yes	No	Number of active pages in the partition
PCTROWCOMP	Yes	Yes	No	Percentage of rows compressed within the total number of active rows inthe partition; -1 to trigger DB2's use of the **default value** (0)

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 Section 2 of *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.CLUSTERRATIO is to 100%, the more closely the ordering of the index entries matches the physical ordering of the table rows. Refer to Figure 130 on page 5-254 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.

Statistics for Created Temporary Tables

When preparing an SQL statement that refers to a 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 temporary table. These values are normally -1 because RUNSTATS cannot run against a 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 temporary table. You can manually update the values in the CARDF and NPAGES columns of the SYSTABLES row for the 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 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.

Using RUNSTATS to 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 86 on page 5-244. For instructions on using RUNSTATS, see Section 2 of *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.

Why Use RUNSTATS: 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 Use RUNSTATS: To ensure that information in the catalog is current, invoke RUNSTATS 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, in order 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 “Is it Necessary to Rebind after Running RUNSTATS?” on page 5-258 for more information.
- 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 use RUNSTATS to collect statistics for a single data partition or index partition. This allows you to avoid the cost of running RUNSTATS against unchanged partitions. When you run RUNSTATS by partition, DB2 uses the results to update the aggregate statistics for the entire table space or index. DB2 cannot calculate the aggregate statistics unless statistics exist for each separate partition. 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.

Recommendation for Performance: To reduce the processor consumption of RUNSTATS 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.

Updating the Catalog

If you have sufficient privileges, you can change all of the values listed in Table 86 on page 5-244 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

There are relationships 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 5-248.

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 129. The range between FIRSTKEYCARDF and FULLKEYCARDF os 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 129. 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 FIRSTKEYCARD 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); so 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 X-78 and “DB2 Codes for Numeric Data” on page X-80. Be sure to allow for a null indicator in keys that allow nulls; see also “Null Values” on page X-77.

Statistics for Uniform 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 Section 2 of *Utility Guide and Reference* for information about collecting more statistics related to columns that are correlated.

You can insert, update, or delete that 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 X-78 and “DB2 Codes for Numeric Data” on page X-80. Be sure to allow for a null indicator in keys that allow nulls; see also “Null Values” on page X-77.

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. We recommend that you update column STATSTIME whenever you update statistics in the catalog, so 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 5-10. Using EXPLAIN to Improve SQL Performance” on page 5-261 for more information about how these statistics are used in access path selection. See Appendix D of *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, CLUSTERRATIO, 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 5-256 and the description of REORG in Section 2 of *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, use RUNSTATS 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. This can ensure that access paths reflect a more "average" state of the data.

This section describes the following topics:

How Clustering affects Access Path Selection

In general, CLUSTERRATIO 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 CLUSTERRATIO 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 CLUSTERRATIO value.

Here are some things to remember about the effect of CLUSTERRATIO on access paths:

- CLUSTERRATIO 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 CLUSTERRATIO value, the lower the cost of referencing data pages during an index scan.
- For an index that has a CLUSTERRATIO less than 80%, sequential prefetch is not used to access the data pages.

Figure 130 shows an index scan on an index with a high cluster ratio. Compare that with Figure 131 on page 5-255, which shows an index scan on an index with a low cluster ratio.

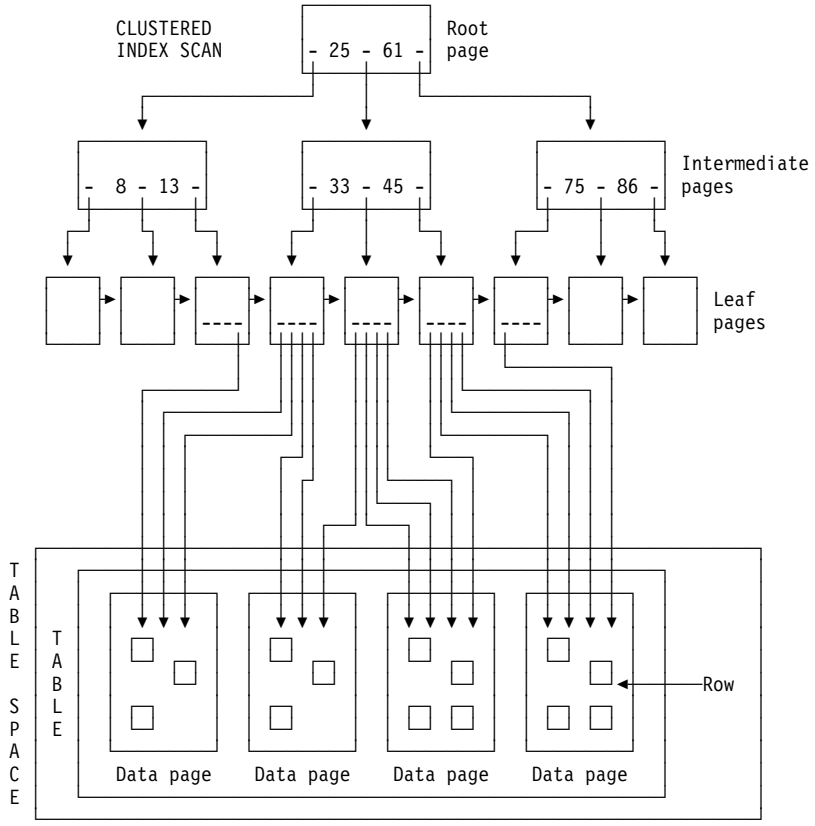


Figure 130. A Clustered Index Scan. This figure assumes that the index is 100% clustered

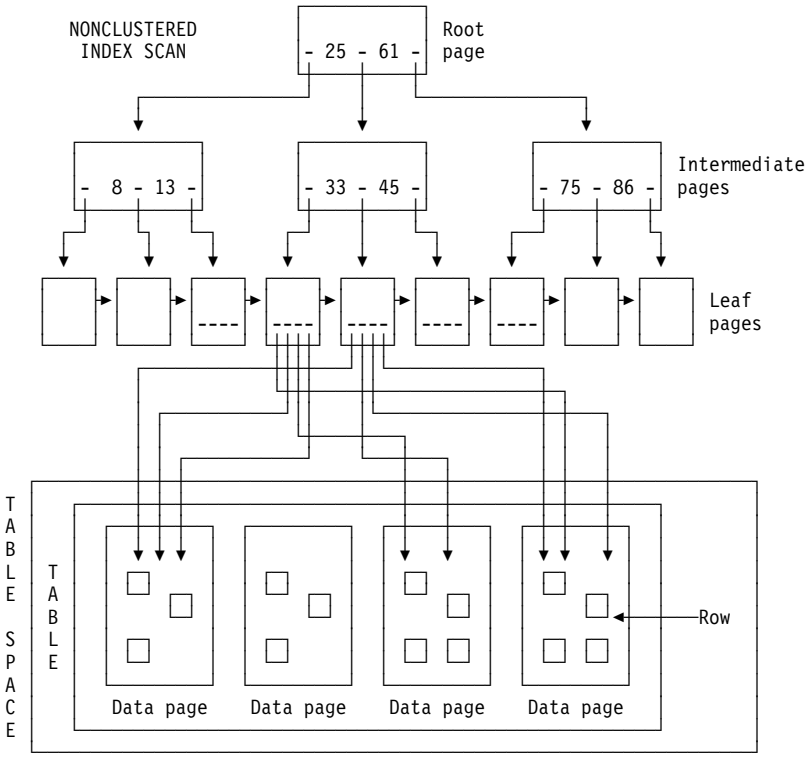


Figure 131. A Nonclustered Index Scan. In some cases, DB2 can access the data pages in order even when a nonclustered index is used.

Other Index-Related Statistics

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, the less the cost.

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, the less the cost.

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. 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.

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, and it can also help reduce the amount of DASD 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 Section 2 of *Utility Guide and Reference* for more information.

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.

Indexes

To understand index organization, you must understand the LEAFDIST column of SYSIBM.SYSINDEXPART. This section describes how to interpret that value and then describes some rules of thumb for determining when to reorganize the index.

How LEAFDIST is Calculated: The LEAFDIST column of SYSIBM.SYSINDEXPART indicates the average number of pages that are between successive leaf pages in the index. Leaf pages can have page gaps whenever index keys are deleted, as shown in Figure 134 on page 5-257 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.

The optimal value of the LEAFDIST catalog column is zero. However, immediately after you run the REORG and RUNSTATS utilities, LEAFDIST might be greater than zero, because of empty pages for FREEPAGE and non-leaf pages.

DB2 determines LEAFDIST by multiplying the average number of pages between consecutive leaf pages by 100. The average number of pages between consecutive leaf pages is equal to the sum of “not-used” pages divided by the total number of leaf pages and then multiplied by 100.

For example, Figure 132 shows 10 leaf pages with 0 pages (gaps) between successive leaf pages. The average number of pages between leaf pages is 0 divided by 10, or 0. So in this case, LEAFDIST is 0.

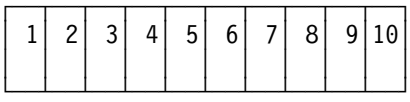


Figure 132. Index in which LEAFDIST=0

However, Figure 133 on page 5-257 shows that if pages 4, 6, 7, and 8 are deleted, there are now 4 pages (gaps) between successive leaf pages. The average number of pages is 4 divided by 6, or 0.66. Using these numbers, the formula for calculating LEAFDIST is 0.66 × 100. So, in this case, LEAFDIST is 66.

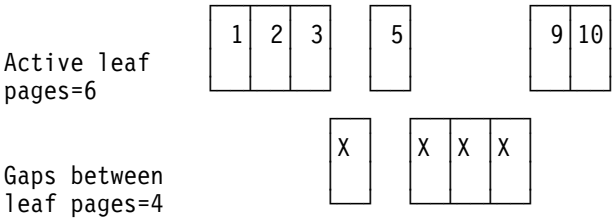
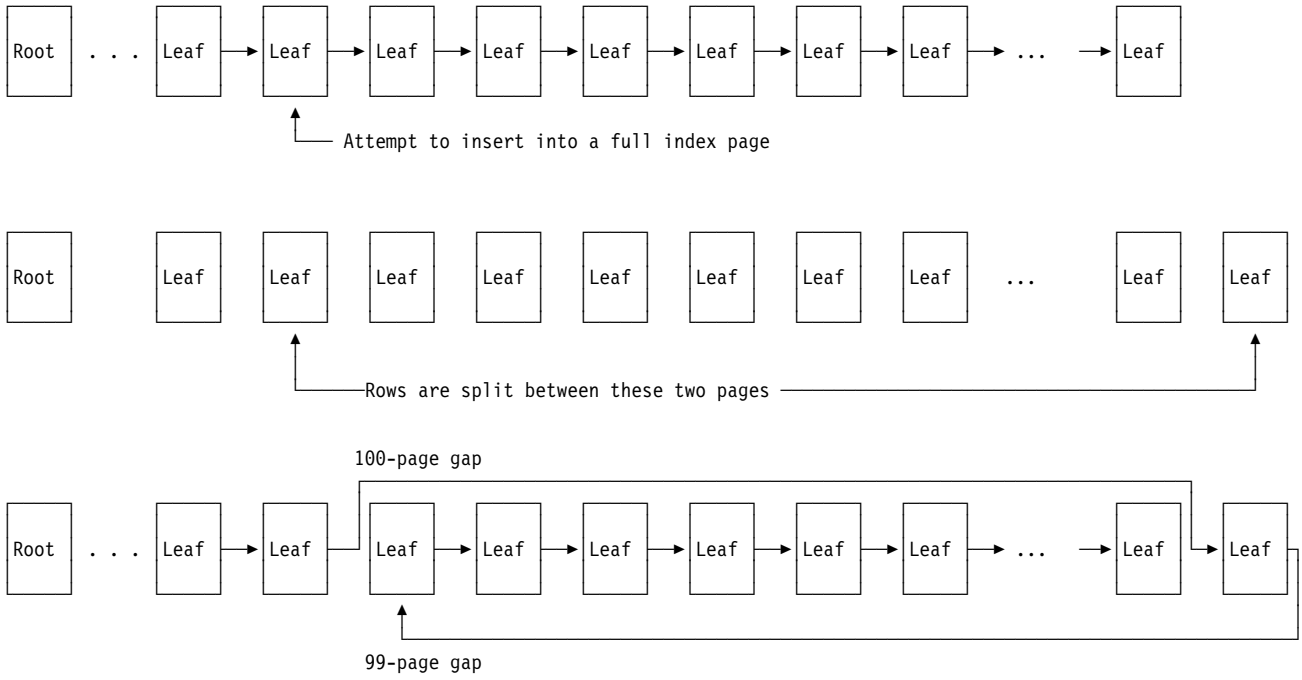


Figure 133. Gaps Caused by Deleted Index Keys

Figure 134 shows that for an index scan which has had a page split, LEAFDIST can be a fairly large value, even though in this case there is only one extra I/O to DASD to find the page that is out of order.




```
103 leaf pages
199 gaps
LEAFDIST = 100 x (199/103) = 193
```

Figure 134. How LEAFDIST is Affected by Index Page Splits. When a key cannot be inserted, DB2 splits the page and puts half of the key values from the original page somewhere else. In this case, it is at the end of the page set. An index scan of the leaf pages must skip ahead to read in the values that were moved.

Rules of Thumb: Consider running REORG INDEX in the following cases:

- LEAFDIST > 200

  Again, as Figure 134 shows, a large LEAFDIST is not always a cause to automatically run REORG. If you don't have free pages, for example, LEAFDIST can grow rapidly. If you have FREEPAGE 0, LEAFDIST can increase by 100 on each split (on average), assuming random distribution.

  If a particular LEAFDIST value seems to correlate with degraded performance for statements on that table space, make that LEAFDIST value your cue to reorganize the index.

- When the data set has multiple extents

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.

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 CLUSTERRATIO column of SYSIBM.SYSINDEXES is less than 90%
- (NEARINDREF + FARINDREF) / CARDF is greater than 10%.
- PERCDROP is greater than 10%
- When the data set has multiple extents

Is it Necessary to Rebind after Running RUNSTATS?

It is not always necessary to rebind all applications after running RUNSTATS. 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:

- CLUSTERRATIO changes to less or more than 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
- NACTIVE 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

In order 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 modelled on those in Figure 135 on page 5-259 to build those statements. Use the information in Table 86 on page 5-244 to generate a complete set of SQL statements.

Notes to Figure 135 on page 5-259:

Product-sensitive Programming Interface

```
SELECT DISTINCT 'UPDATE SYSIBM.SYSTABLESPACE SET NACTIVE='
CONCAT DIGITS(NACTIVE)
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',CLUSTERRATIO='CONCAT DIGITS(CLUSTERRATIO)
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;

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 135. Statements to Generate Update Statistics on Test System

- 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. If the access paths from test to production are different, there are the following possible causes:

- The processor models are different.
- The buffer pool sizes are different.
- There is mismatching data in SYSIBM.SYSCOLDIST. (This is only possible 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 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 5-10. 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 xi.

Definitions and Purpose: EXPLAIN is a monitoring tool that produces information about a plan, package, or SQL statement when it is bound. The output appears in a user-supplied table called PLAN_TABLE, which we refer to as a *plan table*. The information 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 the join method and type, 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 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.

Chapter Overview: This chapter includes the following topics:

DB2 Visual Explain: DB2 Visual Explain is a graphical workstation feature of DB2 Version 5 that is used for analyzing and optimizing DB2 SQL statements. This feature 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

Working from a workstation client, you can display and analyze the PLAN_TABLE output or graphs of access paths chosen by DB2. Visual Explain uses the

PLAN_TABLE to obtain the information that is displayed. The relationships between database objects such as tables or indexes, or operations such as table space scans, sorts, or joins are easier to understand with this graphical view. Visual Explain not only displays the details of the access path of an SQL statement, it also allows you to invoke EXPLAIN for dynamic SQL statements. In some cases, Visual Explain provides suggestions that can enhance the application's or the statement's efficiency or performance. By using Visual Explain, you can access the catalog statistics for certain referenced objects of an access path.

In addition, the subsystem parameters can be displayed, with the current values, ranges of values that are possible, and descriptions of each parameter that is selected. The current values of the subsystem parameters used along with the access path information can provide you with a more complete understanding of your SQL environment. For information on using DB2 Visual Explain, which is a separately packaged CD-ROM provided with your DB2 Version 5 license, see *DB2 Visual Explain online help*.

An Alternative Tool: DB2 Performance Monitor (PM) for Version 5 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.

Obtaining Information from EXPLAIN

To obtain information to interpret, you must:

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 5-267.
3. Select the information you want from the table. For instructions, see “Reordering Rows from a Plan Table” on page 5-269.

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.

Figure 136 on page 5-263 shows the format of a plan table. Table 87 on page 5-263 shows the content of each column.

Your plan table can use the 25-column format, the 28-column format, the 30-column format, the 34-column format, the 43-column format, or the 46-column format. We recommend the 46-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 136 on page 5-263, certain columns do allow nulls. Do not specify those columns as NOT NULL WITH DEFAULT.

```
QUERYNO           INTEGER       NOT NULL          PREFETCH          CHAR(1)        NOT NULL
QBLOCKNO          SMALLINT      NOT NULL          COLUMN_FN_EVAL    CHAR(1)        NOT NULL
APPLNAME          CHAR(8)       NOT NULL          MIXOPSEQ          SMALLINT       NOT NULL
PROGNAME          CHAR(8)       NOT NULL              -------28 column format --------
PLANNO            SMALLINT      NOT NULL          VERSION           VARCHAR(64)    NOT NULL
METHOD            SMALLINT      NOT NULL          COLLID            CHAR(18)       NOT NULL
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
SORTC_UNIQ        CHAR(1)       NOT NULL          JOIN_TYPE         CHAR(1)        NOT NULL
SORTC_JOIN        CHAR(1)       NOT NULL          GROUP_MEMBER      CHAR(8)        NOT NULL
SORTC_ORDERBY     CHAR(1)       NOT NULL          IBM_SERVICE_DATA  VARCHAR(254)   NOT NULL
SORTC_GROUPBY     CHAR(1)       NOT NULL              ------43 column format --------
TSLOCKMODE        CHAR(3)       NOT NULL          WHEN_OPTIMIZE     CHAR(1)        NOT NULL
TIMESTAMP         CHAR(16)      NOT NULL          QBLOCK_TYPE       CHAR(6)        NOT NULL
REMARKS           VARCHAR(254)  NOT NULL          BIND_TIME         TIMESTAMP      NOT NULL;
    -------25 column format --------                 ------46 column format -----------
```

Figure 136. Format of PLAN_TABLE

Table 87 (Page 1 of 5). 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, you can specify the number in the SET QUERYNO clause; otherwise, DB2 assigns a number based on the line number of the SQL statement in the source program. FETCH statements do not each have an individual QUERYNO assigned to them. Instead, DB2 uses the QUERYNO of the DECLARE CURSOR statement for all corresponding FETCH statements for that cursor. Values of QUERYNO 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.

Table 87 (Page 2 of 5). Descriptions of Columns in PLAN_TABLE

Column Name	Description
METHOD	A number (0, 1, 2, 3, or 4) that indicates the join method used for the step: 0 First table accessed, continuation of previous table accessed, or not used. 1 *Nested loop* join. For each row of the present composite table, matching rows of a new table are found and joined. 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. 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. 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, temporary table, materialized view, table expression, or an intermediate result table for an outer join that is accessed in this step; blank if METHOD is 3. For an outer join, this column contains the temporary table name of the work file in the form DSNWFQB(*qblockno*). Merged views show the base table names and correlation names. A materialized view is another query block with its own materialized views, tables, and so forth.
TABNO	Values are for IBM use only.
ACCESSTYPE	The method of accessing the new table: I By an index (identified in ACCESSCREATOR and ACCESSNAME) I1 By a one-fetch index scan N By an index scan when the matching predicate contains the IN keyword R By a table space scan M By a multiple index scan; followed by MX, MI, or MU MX By an index scan on the index named in ACCESSNAME MI By an intersection of multiple indexes MU By a union of multiple indexes 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 5-272.
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.

Table 87 (Page 3 of 5). Descriptions of Columns in PLAN_TABLE

Column Name	Description
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: IS Intent share lock IX Intent exclusive lock S Share lock U Update lock X Exclusive lock SIX Share with intent exclusive lock N UR isolation; no lock 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. NS For UR isolation, no lock; for CS, RS, or RR, an S lock. NIS For UR isolation, no lock; for CS, RS, or RR, an IS lock. NSS For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock. SS For UR, CS, or RS isolation, an IS lock; for RR, an S lock. 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.
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. 1, 2, ... n For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.) 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, and can be 0 if there is a host variable. The actual number of parallel operations used at execution time could be different. This column contains 0 if there is a host variable.

Table 87 (Page 4 of 5). Descriptions of Columns in PLAN_TABLE

Column Name	Description
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; I Query I/O parallelism C Query CP parallelism 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 an outer join. F FULL OUTER JOIN L LEFT OUTER JOIN blank INNER JOIN or no join 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.
WHEN_OPTIMIZE	When the access path was determined: blank At bind time, using a default filter factor for any host variables, parameter markers, or special registers. B At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however the statement will be 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. 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.

Table 87 (Page 5 of 5). Descriptions of Columns in PLAN_TABLE

Column Name	Description
QBLOCK_TYPE	For each query block, the type of SQL operation performed. For the outermost query, it identifies the statement type. Possible values: SELECT SELECT INSERT INSERT UPDATE UPDATE DELETE DELETE SELUPD SELECT with FOR UPDATE OF DELCUR DELETE WHERE CURRENT OF CURSOR UPDCUR UPDATE WHERE CURRENT OF CURSOR CORSUB Correlated subquery NCOSUB Noncorrelated subquery
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.

Populating and Maintaining a Plan Table

For the two distinct ways to populate a plan table, see:

- "Execute the SQL Statement EXPLAIN"
- "Bind with the Option EXPLAIN(YES)"

For a variation on the first way, see "Executing EXPLAIN Under QMF" on page 5-268.

For tips on maintaining a growing plan table, see "Maintaining a Plan Table" on page 5-269.

Execute 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 *SQL Reference*.

Bind 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 5-215.) 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 5-224.

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;
```

First 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)"
- "Is Access Through More than One Index? (ACCESSTYPE is M, MX, MI, or MU)" on page 5-271
- "How Many Columns of the Index Are Used in Matching? (ACCESSTYPE is I, I1, N, or MX)" on page 5-272
- "Is the Query Satisfied Using Only the Index? (INDEXONLY=Y)" on page 5-272
- "Is a View Materialized into a Work File? (TNAME names a view)" on page 5-272
- "Was a Scan Limited to Certain Partitions? (PAGE_RANGE=Y)" on page 5-273
- "What Kind of Prefetching Is Done? (PREFETCH is L, S, or blank)" on page 5-273
- "Is Data Accessed or Processed in Parallel? (PARALLELISM_MODE is I, C, or X)" on page 5-274
- "Are Sorts Performed?" on page 5-274
- "Is a Subquery Transformed into a Join? (QBLOCKNO Value)" on page 5-274
- "When Are Column Functions Evaluated?" on page 5-275

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)

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 5-278.

The plan table does not identify whether the index is type 1 or type 2. To determine that, query the column INDEXTYPE in the catalog table SYSIBM.SYSINDEXES: the value is 2 for a type 2 index or blank for a type 1 index.

Is Access Through More than One Index? (ACCESSTYPE is M, MX, MI, or MU)

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 137 and Figure 138 have these indexes: IX1 on T(C1) and IX2 on T(C2). DB2 processes the query in these steps:

1. Retrieve all the qualifying row 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 137. 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 138.

```
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 138. PLAN_TABLE Output for Example with Union (OR) Operator

The steps are:

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? (ACCESSTYPE is I, I1, N, or MX)

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. There are two equal predicates on the first two columns and a range predicate on the third column. Though there are 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
(described under “What Kind of Prefetching Is Done? (PREFETCH is L, S, or blank)” on page 5-273. Index-only access is not possible for varying-length data,
unless the RETVLCFK subsystem parameter is set to YES. See Section 2 of
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.

Is a View Materialized into a Work File? (TNAME names a view)

TNAME names a view.

A view is *materialized* if the data rows it selects are put into a work file to be processed like a table. If a view in your query is materialized, that step appears in the plan table with a separate value of QBLOCKNO and the name of the view in TNAME. When DB2 can process the view by referring only to the base table, there is no view name in the column TNAME. (For a more detailed description of view materialization, see “View Processing” on page 5-296.)

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'
   AND 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 either 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.
- A matching index scan on column C2. (DB2 might choose that if few rows have C2=6.) The matching index scan alone reads all RIDs for C2=6 from the index on C2 and all corresponding data pages.
- For a table space scan, DB2 avoids reading data pages from any partitions except 3, 4, 8 and 9.

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 is 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 5-291.
- L, the method is called *list sequential 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 5-289) when the value of column SORTN_JOIN is N. For a more

complete description, see “List Sequential Prefetch (PREFETCH=L)” on page 5-291.

- 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, see “Sequential Detection at Execution Time” on page 5-292.

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 was planned. For a more complete description , see “Parallel Operations and Query Performance” on page 5-299.

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. 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? (QBLOCKNO Value)

A plan table shows that a subquery is transformed into a join by the value in column QBLOCKNO. If the subquery is executed in a separate operation, 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?

When the column functions (SUM, AVG, MAX, MIN, COUNT) 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.

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. The topics are:

- Table Space Scans (ACCESSTYPE=R PREFETCH=S)
- "Overview of Index Access" on page 5-276
- "Index Access Paths" on page 5-278
- "UPDATE Using an Index" on page 5-282

Table Space Scans (ACCESSTYPE=R PREFETCH=S)

Table space scan is most often used for one of the following reasons:

- Access is through a temporary table. (Index access is not possible for temporary tables.)
- A matching index scan is not possible because an index is not available, or there are no predicates to 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 in order 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 88 summarizes the recommendations for SEGSIZE, depending on how large the table is.

Table 88. 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 5-273.

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 5-291.

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

Both type 1 and type 2 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 sequential 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 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 “Using OPTIMIZE FOR n ROWS” on page 5-234.)

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 there are operating system restrictions on the number of open data sets.
- If you have concurrent users of the same table, locking problems are likely with multiple type 1 indexes. Type 2 indexes can sometimes give high concurrency and better performance by locking the underlying data page or record instead of locking the index page.
- 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, especially for type 1 indexes with many duplicate values, either for particular keys or across the whole key range. A type 2 index is the better choice.

  Because the RIDs are not ordered, to delete (or update) one type 1 index entry from a set of duplicates requires more processing than to delete a unique entry.

  With a type 2 nonunique index, searching a long chain of duplicate key values to locate a specific key to be deleted from the index is very efficient because the RIDs are kept in order.
- Indexes can be built automatically when loading data, but this takes time. They must be recovered if the underlying table space is recovered, which is also time consuming.

The costs to be considered for indexes are relevant for type 1 or type 2 indexes. However, it is a general recommendation that you make all your indexes type 2. Type 2 indexes can reduce some concurrency problems by their use of data-only locking. Many other functions in DB2, such as CP-parallelism and row locking, cannot be used without type 2 indexes.

Recommendation: In reviewing the access paths described in the next section, consider indexes as part of your database design, See “ Section 2. Designing a Database” on page 2-1 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)"
- "Index Screening" on page 5-279
- "Nonmatching Index Scan (ACCESSTYPE=I and MATCHCOLS=0)" on page 5-279
- "IN-list Index Scan (ACCESSTYPE=N)" on page 5-279
- "Multiple Index Access (ACCESSTYPE is M, MX, MI, or MU)" on page 5-280
- "One-Fetch Access (ACCESSTYPE=I1)" on page 5-281
- "Index-only Access (INDEXONLY=Y)" on page 5-282
- "Equal Unique Index (MATCHCOLS=number of index columns)" on page 5-282

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, if there is at least one indexable Boolean term predicate on that column, it is a match column. (See "Properties of Predicates" on page 5-206 for a definition of Boolean term.)

  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.

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.

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;
```

There are two matching columns 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):

```
SELECT * FROM T
  WHERE C1 = 1
   AND C3 > 0 AND C4 = 2
   AND C5 = 8;
```

C3>0 and C4=2 are index screening predicates. They 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.

The index is not screened when list prefetch is used or during the MX phases of multiple index access. EXPLAIN does not directly tell when an index is screened; however, you can tell from the query, the indexes used, and the value of MATCHCOLS in the plan table.

Nonmatching Index Scan (ACCESSTYPE=I and MATCHCOLS=0)

In a *nonmatching index scan* there are no matching columns in the index. Hence, all the index keys must be examined.

Because a nonmatching index usually provides no filtering, there are only a few cases when it is an efficient access path. The following situations are examples:

- When there are index screening predicates

  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 there is more than one table in a nonsegmented table space

  In that case, a table space scan reads irrelevant rows. By accessing the rows through the nonmatching index, there are fewer rows to 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)
```

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, the values of ACCESSTYPE are:

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 139 on page 5-281:

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 139. 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.
- There is an ascending index column for MIN, and a descending index column for MAX.
- 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: The following queries use one-fetch index scan with an index existing on `T(C1,C2 DESC,C3)`:

```
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 MAX(C2) FROM T WHERE C1=5;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2>5;
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.)

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

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 5-283, 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 5-285
- “Merge Scan Join (METHOD=2)” on page 5-287
- “Hybrid Join (METHOD=4)” on page 5-289

Definitions and Examples

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 140 on page 5-284 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:

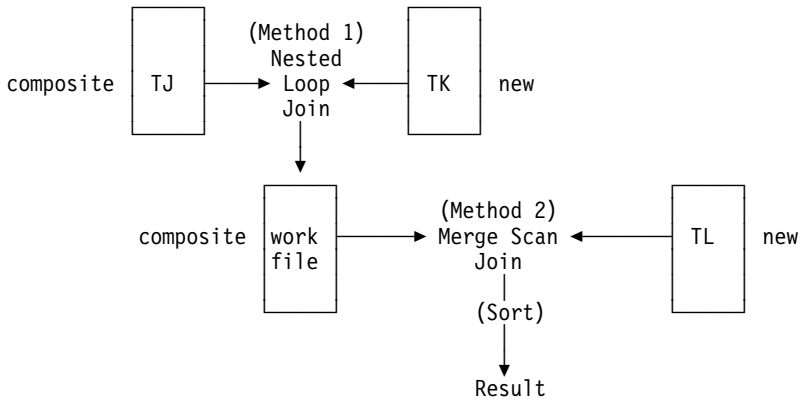
- An *inner join* discards rows of either table that do not match any row of the other table.
- An *outer join* keeps unmatched rows of one or the other table, or of both. A row in the composite table that results from an unmatched row is filled out with null values. Outer joins are distinguished by which unmatched rows they keep.

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 141 on page 5-284 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, so JOIN_TYPE never identifies a right outer join specifically.



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 140. Join Methods as Displayed in a Plan Table

```
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 141. Plan Table Output for an Example with Outer Joins

Table Names for Work Files: DB2 creates a temporary work file for subquery with several joins, at least one of which is an outer join. If you do not specify a correlation name, DB2 gives the work file a name, which appears in column TNAME of the plan table. The name of the work file is 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.

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 5-208.) 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 142 on page 5-286 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.

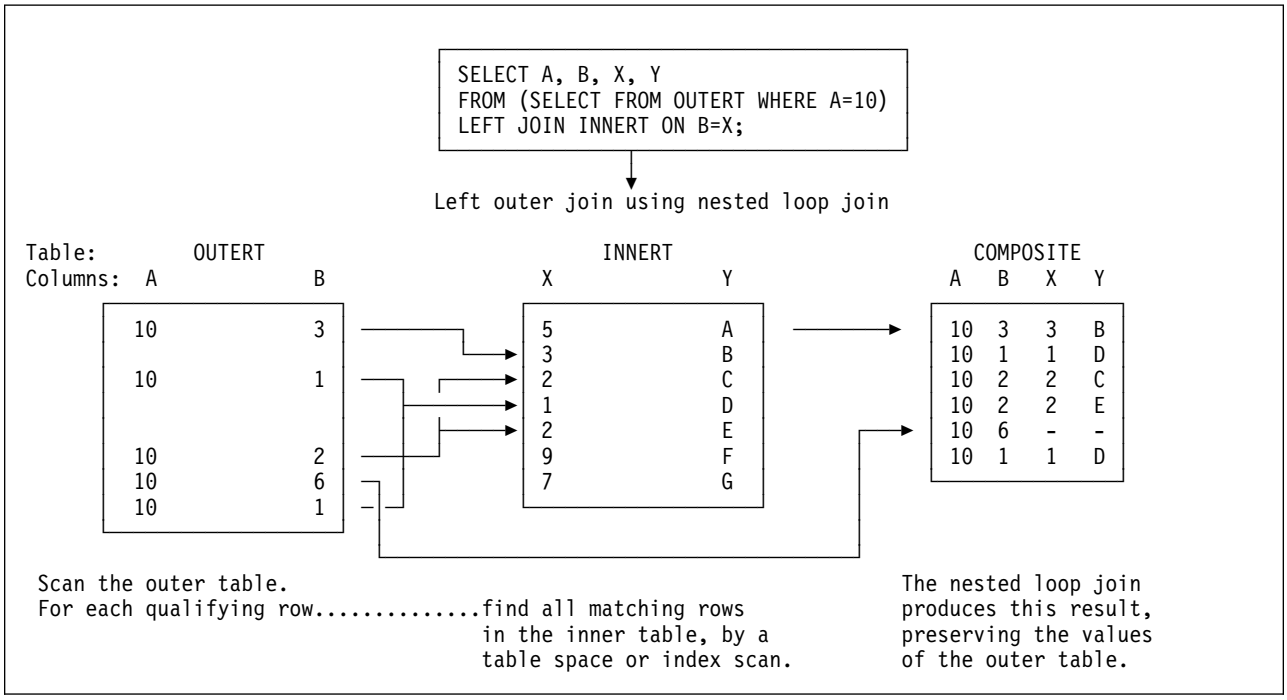


Figure 142. Nested Loop Join for a Left Outer Join

```
SELECT * FROM T1, T2, T3
   WHERE  T1.C1 = T3.C1 AND
          T2.C2 = T3.C2 AND
          T3.C3 = 5;
```

There are join predicates 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 142) if:

- The join columns in the composite table and the new table are not in the same sequence.
- There is no index on the join column of the composite table.
- There is an index, but it 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 143 on page 5-288 illustrates a 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:
  - 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:
    - 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.

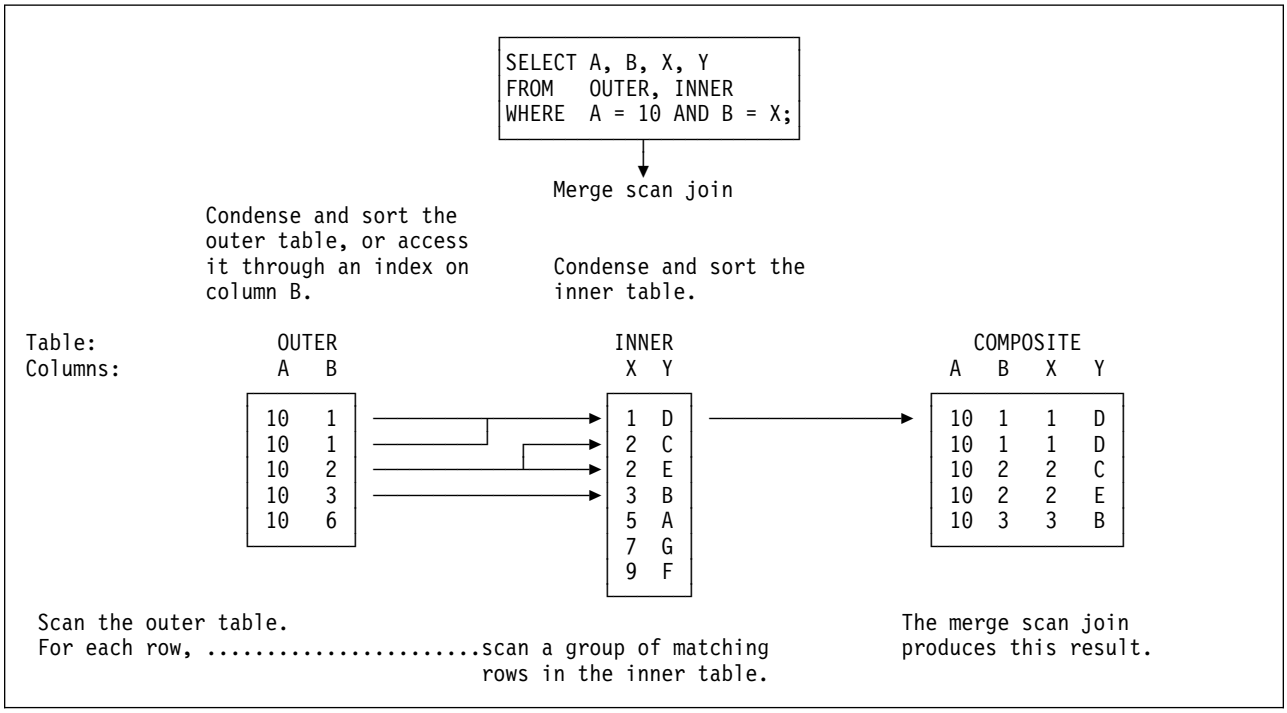


Figure 143. Merge Scan Join

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.

Hybrid Join (METHOD=4)

The method applies only to an inner join and requires an index on the join column of the inner table.

Method of Joining

The method requires obtaining RIDs in the order needed to use list prefetch. The steps are shown in Figure 144 on page 5-290. 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.

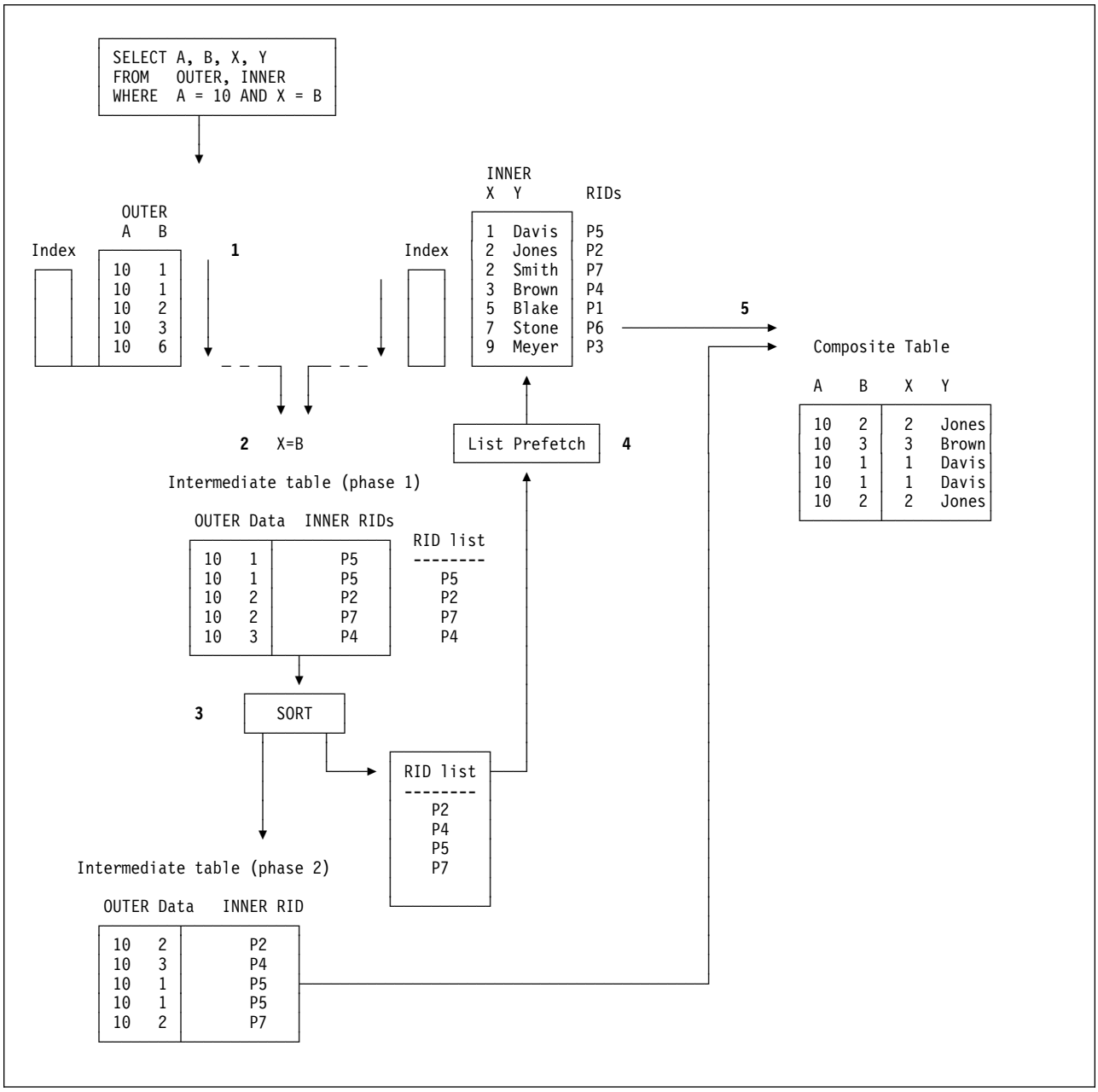


Figure 144. Hybrid Join (SORTN_JOIN='Y')

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
- There are duplicate qualifying rows in the outer table

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 Sequential Prefetch (PREFETCH=L)"

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 5-292

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 4KB buffer pools

Buffer Pool Size	Pages Read by Prefetch
<=223 buffers	8 pages for each asynchronous I/O
224-999 buffers	16 pages for each asynchronous I/O
1000+ buffers	32 pages for each asynchronous I/O

- For 32KB buffer pools

Buffer Pool Size	Pages Read by Prefetch
<=16 buffers	0 pages (prefetch disabled)
17-99 buffers	2 pages for each asynchronous I/O
100+ buffers	4 pages for each asynchronous I/O

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 Sequential Prefetch (PREFETCH=L)

List sequential prefetch reads a set of data pages determined by a list of RIDs taken from an index. The data pages need not be contiguous. List prefetch can be used in conjunction with either single or multiple index access.

The Access Method

The three steps in list sequential prefetch are:

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 sequential 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 sequential 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 sequential prefetch is used.

When It Is Used

List sequential 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 sequential 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 5-69.

During execution, DB2 ends list sequential 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 sequential 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 there is no remaining step, 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.

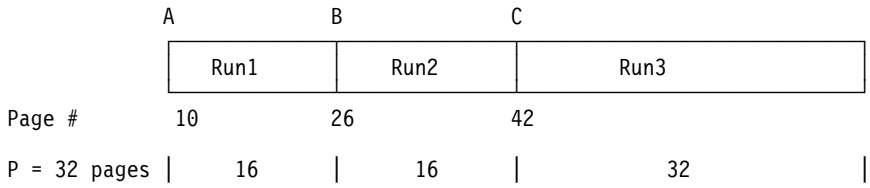


Figure 145. Initial Page Ranges to Determine When to Prefetch

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 5-53 for a description of these thresholds.

Determining Sort Activity

There are 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 there is no index on the GROUP BY column.

Sorts to Remove Duplicates

This type of sort is used for 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

DB2 sorts RIDs into ascending page number order in order to perform list prefetch. 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.

View Processing

There are two methods used to satisfy your queries that reference views:

- *View merge*
- *View materialization*

You can determine the methods used by executing EXPLAIN for the referencing view statement. The following information helps you understand your EXPLAIN output about views, and helps you tune your queries that reference views.

View Merge

In the view merge process, the statement that references the view is combined with the subselect that defined the view. This combination creates a logically equivalent statement. This equivalent statement is executed against the database. Consider the following statements:

```
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 subselect 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);
```

View Materialization

Views cannot always be merged. In 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 subselect, 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 View Materialization

In the previous example, DB2 performs view materialization, which is a two step process.

1. The view's defining subselect is executed against the database and the results are placed in a temporary copy of a result table.
2. The view's referencing statement is then executed against the temporary copy of the result table to obtain the intended result.

Whether a view needs to be materialized depends upon the attributes of the view referencing statement, or logically equivalent referencing statement from a prior view merge, and the view's defining subselect.

When Views or Nested Table Expressions are Materialized

In general, DB2 uses materialization to satisfy a reference to a view or nested table expression when there is aggregate processing (grouping, column functions, distinct), indicated by the defining subselect, in conjunction with either aggregate processing indicated by the statement referencing the view or nested table expression, or by the view or nested table expression participating in a join.Table 89 indicates some cases in which materialization occurs. DB2 can also use materialization in statements that contain multiple outer joins, or outer joins combined with inner joins.

Table 89. 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 nested table expression uses...(1)	**View definition or nested table expression uses...(2)**				
	GROUP BY	DISTINCT	Column function	Column function DISTINCT	Outer join
Inner join	X	X	X	X	X
Outer join (3)	X	X	X	X	X
GROUP BY	X	X	X	X	X
DISTINCT	-	X	-	X	-
Column function	X	X	X	X	X
Column function DISTINCT	X	X	X	X	X
SELECT subset of view or nested table expression columns	-	X	-	-	-

Notes to Table 89:

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 6 of *SQL Reference* for information on which views are read-only (not updateable).

   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 nested table expression, then materialization occurs. For example:

```
SELECT C1 FROM
   (SELECT :HV1 AS C1 FROM T1) X;
```

3. Additional details about materialization with outer joins:

   - If a WHERE clause exists in a view or nested 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;
     ```

   - Most cases of nested outer join statements cause views and nested table expressions to be materialized.
   - If the result of an outer join undergoes another join of any type, the result of the first outer join is materialized before the next join begins.
   - If the result of an inner join undergoes a further outer join, the result of the first inner join is materialized before the next join begins.

Using EXPLAIN to Determine the View Method

For each reference to a view 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 subselect, and they describe the access to the temporary result as indicated by the view referencing statement. The defining subselect can also reference views that need to be materialized.

Another indication that DB2 chose view materialization is that the view name appears as a TNAME attribute for rows describing the access path for the view referencing query block. When DB2 chooses view merge, EXPLAIN data for the merged statement appears in PLAN_TABLE; only the names of the base tables on which the view is defined appear.

Performance of View Methods

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, view 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 view 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 90.

Table 90. 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.

Performance of Table Expressions

A table expression is the specification of a subquery in the FROM clause of an SQL SELECT statement. This subquery can be used as an operand of an outer join operation. The table expression is similar to a view in that it can be merged or materialized. A table expression is merged after a view merge. The following simple example of a table expression uses “TX” as the expression for the subquery:

```
SELECT * FROM
   (SELECT C1 FROM T1) AS TX;
```

See Table 89 on page 5-297 for the cases when table expressions are materialized.

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 *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.

Partitioned vs. Nonpartitioned Table Spaces: Although partitioned table spaces show the most performance improvements, nonpartitioned table spaces might benefit in processor-intensive queries:

- For a merge scan join, the join phase can be processed in parallel because the sort work files can be partitioned before performing the join.

  The partitioning of the work files is possible only if the hardware sort facility is available at run time.

- In the nested loop join, DB2 is more likely to choose parallelism if the outer table is partitioned.

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 146 on page 5-301 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 146 on page 5-301, a prefetch request takes longer than the time to process it. The processor is frequently waiting for I/O.

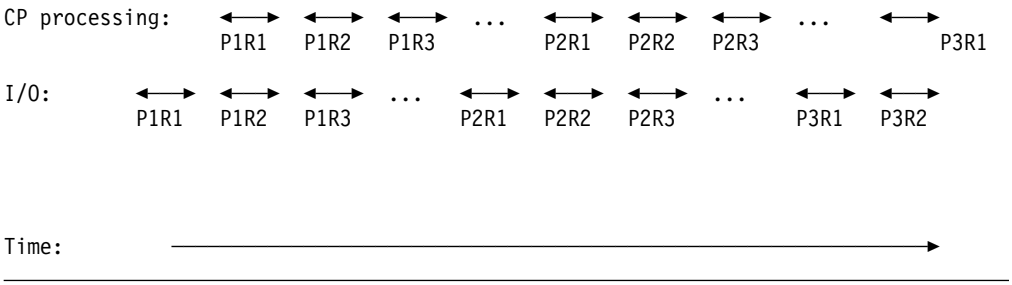


Figure 146. CP and I/O Processing Techniques. Sequential processing.

Figure 147 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.

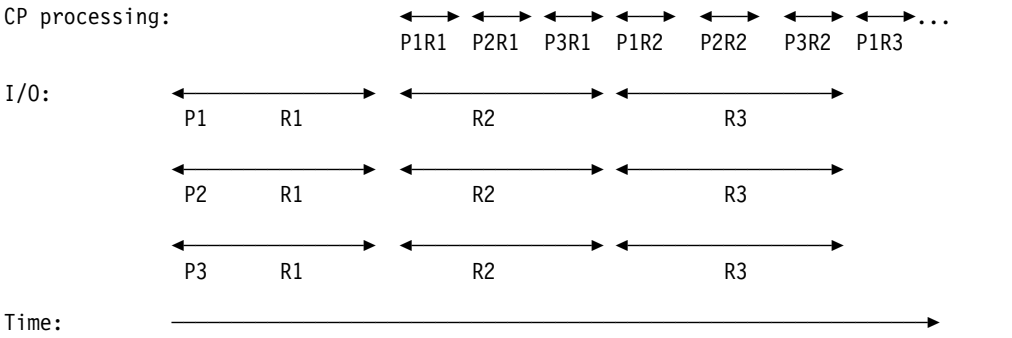


Figure 147. CP and I/O Processing Techniques. Parallel I/O processing.

Figure 148 on page 5-302 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.

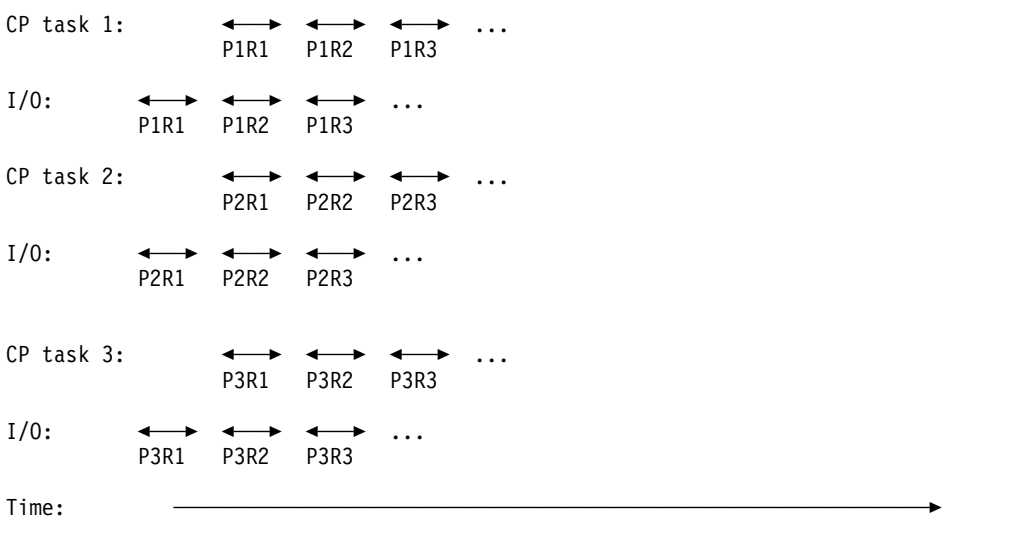


Figure 148. CP and I/O Processing Techniques. Query processing using CP parallelism. The tasks can be contained within a single CPC or can be spread out among the members of a data sharing group.

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 most of the time using processor 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. When using parallel query CP processing or Sysplex query parallelism, a task is an actual MVS execution unit used to process a query. When using 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 set of parallel operations. 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 primary agent that receives data from multiple execution units (referred to as **parallel tasks**). The originating task controls the creation of the parallel tasks and maintains the status of each parallel task.

Partitioning for Optimal Parallel Performance

In this section, we describe 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. This number is 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 rules of thumb. 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 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 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 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. (See “Example Configurations for an I/O-Intensive Query” on page 5-305 for more about I/O path configurations.) 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 149 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.[12]

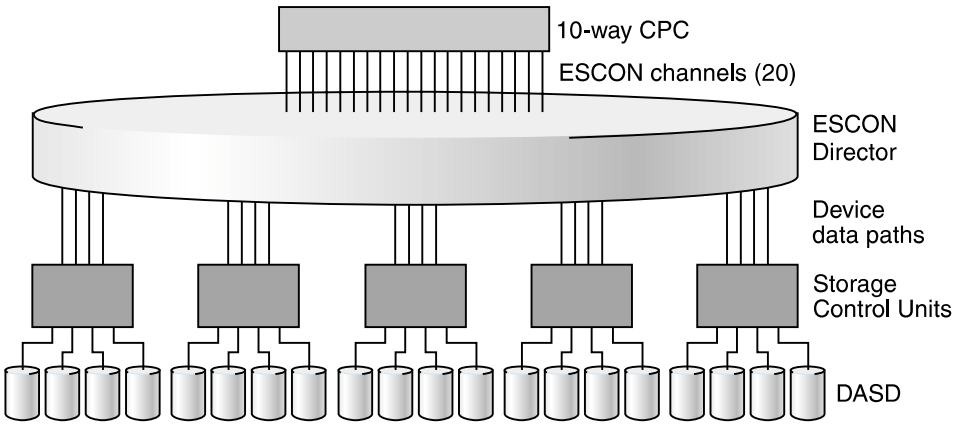


Figure 149. I/O Configuration that Maximizes Performance for an I/O-Intensive Query

What if the Table Space 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.

Make 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

[12] 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.

| has evenly sized partitions of 1000 pages each, and the query is I/O-intensive, then
| ten logical work ranges might be created. This would result in a degree of
| parallelism of 10 and reduced elapsed time.

Because logical partition sizes are based on what the catalog says their size is, the
catalog should be up to date.

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.

Enabling Parallel Processing

Queries can only take advantage of parallelism if you enable parallel processing. 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.
- 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 5-53.
- # 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.

If you enable parallel processing, when DB2 estimates a given query's I/O and central processor cost is high, it can activate multiple parallel tasks if it 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. Also needed are functions available in MVS/ESA Version 5 Release 2 or subsequent releases. Without these, 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 91 on page 5-307.

Limiting the degree of parallelism: You can limit the degree of parallelism so that DB2 doesn't create too many parallel tasks taking virtual storage. To limit the degree of parallelism, change the subsystem parameter PARAMDEG in the DSN6SPRM macro in DSNTIJUZ from the default 0 to a different value. DSNTIJUZ must be assembled and link edited and DATABASE 2 subsystem must be stopped and started to make the change effective. To disable parallelism, see "Disabling Query Parallelism" on page 5-313.

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 91 to see if your query uses any of the access paths that do not allow parallelism.

Table 91. 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.
Access through a type 1 index.	Yes	No	No	
Correlated subquery	No	No	No	There is virtually no benefit in using parallelism on the correlated subquery. DB2 tries to run the outer query in parallel. (For noncorrelated queries, DB2 tries to run both the inner and outer queries in parallel.)
IN-list index access	No	No	No	Indicated by N in the ACCESSTYPE column of PLAN_TABLE.
Outer join	No	No	No	Indicated by an F or L in the JOIN_TYPE column of PLAN_TABLE.
Merge scan join on more than one column	No	No	No	
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'.

Effect of Nonpartitioning Indexes: DB2 does not use parallelism when there is direct index access through a nonpartitioning index to the first base table in a parallel group.

Interpreting EXPLAIN Output

To understand how DB2 uses parallel operations, and how the contents of the PLAN_TABLE columns relate to these parallel operations, consider the following examples. The columns mentioned in these examples are described in Table 87 on page 5-263.

All steps 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. For a complete description of join methods, see "Interpreting Access to Two or More Tables" on page 5-282. For each of these examples you could have data in the column PARALLELISM_MODE. This tells you the kind of parallelism that is doing the processing. Within a query block (QBLOCKNO column of PLAN_TABLE), 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.

For these examples, the other values would not change whether the PARALLELISM_MODE 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 MVS/ESA sort
- Ambiguous cursors

| 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 *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 accounting trace records are created for each parallel task, and there is no accumulation of trace record fields. If this generates too many trace records, you can choose to have DB2 roll up those parallel task accounting records into a single accounting record that is cut by the originating task. The originating task's record is still separate from those of the parallel tasks.

To roll up records, add the subsystem parameter PTASKROL to the DSN6SYSP macro invocation in the edited version of your DSNTIJUZ (in *prefix*.NEW.SDSNSAMP) as shown here:

```
DSN6SYSP   AUDITST=NO,
           BACKODUR=5,
              ...
           PCLOSET=10,
           PTASKROL=YES,
             ...
```

If you add PTASKROL before the end of the invokation, add a continuation marker in column 72. Reassemble and re-linkedit DSNTIJUZ, and stop and restart DB2 to make the change effective.

DB2 PM summarizes all accounting records generated for a parallel CP 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 150 CPU TIME-TCB 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
CPU TIME         1:29.602498     1:29.554733    SER.TASK SWTCH      0.000000
 TCB                0.132351        0.088834    SYNCHRON. I/O       0.142382
 TCB-STPROC         0.092802        0.089294    OTHER READ I/O   3:00.404769
 PAR.TASKS     A 1:29.470147     1:29.465898    OTHER WRTE I/O      0.000000

⋮
...            QUERY PARALLEL.     TOTAL
               ---------------  --------
               MAXIMUM MEMBERS         1
               MAXIMUM DEGREE         10
               GROUPS EXECUTED         1
               RAS 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.   G     0
               MEMB SKIPPED(%)   H     0
               DISABLED BY RLF   I   NO
```

Figure 150. 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 7 of *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 150 on page 5-311), check the following factors:

- Buffer pool availability

  Depending on buffer pool availability, DB2 could reduce the degree of parallelism (see **C** in Figure 150 on page 5-311) 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**. If MVS Version 5 Release 2 is not available, you see a value in **G**.

Locking Considerations for Repeatable Read Applications: When using 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 would 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. So, 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), you will see a non-zero value in field QXRLFDPA in IFCID 0002 or 0003 (shown in **I** in Figure 150 on page 5-311). 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 “What the RLST Contains” on page 5-80.

Chapter 5-11. Monitoring and Tuning in a Distributed Environment

DB2 can access distributed data. If requesters and servers support two-phase commit and Distributed Relational Database Architecture (DRDA), your DB2 subsystem can read from and update many locations in a single unit of work, regardless of where your application originates. We call this level of capability *distributed unit of work*.

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*. DB2 chooses between the two connection types based on the SQL statements contained in the application process.

DB2 Private Protocol Access: For distributed work between the two subsystems, DB2 uses communications connections that are specific to DB2. Access using DB2 private protocol has these characteristics:

- Only DB2 subsystems can communicate using this connection.
- An application is not limited to a single location per unit of work.
- An application can direct a query to another DB2 subsystem by using an alias or a three-part name. DB2 determines the remote location from the object name in the query and connects to that subsystem.
- 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.
- Updates at a server DB2 can be made from TSO/BATCH, CAF, IMS, CICS or the Recoverable Resource Manager Services attachment facility (RRSAF)

DRDA Access: For more information on DRDA, see *Distributed Relational Database Architecture: Connectivity Guide*.

With access using DRDA, an application can explicitly connect with another non-DB2 database management and remotely bind and execute packages of static or dynamic SQL that have previously been bound at that location. (A package is a single, bound, database request module.) Distributed processing using DRDA has the following characteristics:

- The application is not restricted to accessing data only at DB2 subsystems.
- An application is not limited to a single location per unit of work.
- The application can use remote BIND to bind SQL into packages at the serving relational database management system.

- The application can use the SQL CONNECT statement to connect to other relational database management systems in the network and execute packages at those database management systems.
- Applications running in TSO/Batch, CAF, IMS, CICS, or RRSAF can use DRDA.
- Updates at a server can be made from TSO/Batch, CAF, IMS, CICS, or RRSAF.

Considerations for 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 negotiating session limits (change number of sessions processing)

Some aspects of overhead processing, for instance, network processing, are not under DB2 control. (You can, however, tune VTAM to improve the performance of your network. For more information on VTAM, refer to Section 3 of *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 system on which the application resides
- The serving system or server
- The connection between the requesting system and the server

The Application and the Requesting System: This includes the application making the distributed request and that part of the database management system that handles distributed processing.

Minimizing the number of messages sent between the requester and the server is a primary way to improve performance. The following measures help reduce message traffic:

- Use the RELEASE SQL statement to release remote connections that are no longer needed. This saves the resources that are required to maintain remote connections. The RELEASE statement does not close cursors, release any resources, or prevent further use of the connection.

  However, when the application issues the RELEASE statement, the database access thread is terminated during commit processing. If the application connects to the same location again, a new database access thread must be created at the server.
- Use DEFER(PREPARE). Using DEFER(PREPARE) can 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 Section 4 of *Application Programming and SQL Guide* .

- Use FOR FETCH ONLY on SELECT statements. See “Using FOR FETCH ONLY to Ensure Block Fetch” on page 5-319 for more information.
- Bind application plans and packages with ISOLATION(CS) whenever possible, which can reduce contention and message overhead.
- If possible, avoid using parameter markers in dynamic SELECT statements.
- Consider using the BIND or REBIND DISCONNECT option to release remote connections if the application is not expected to reuse remote connections after commit. If you are going to release a connection, issue the SQL RELEASE before committing your transaction. This reduces network message traffic.
- Avoid using several SQL statements when one 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. This creates only one send and receive operation (for the CALL statement) instead of a potential send and receive operation for each SQL statement.

  This can significantly lower your processor and elapsed time costs. The greater the number of SQL statements in your application, the greater the performance improvement. For more information on how to use stored procedures, see Section 6 of *Application Programming and SQL Guide*.
- Do not request more columns or rows than you need. Also, use the OPTIMIZE FOR *n* ROWS clause to reduce the number of rows that are returned to the application in each cursor block. See *Application Programming and SQL Guide* for more information about using this clause in distributed applications.
- Consider carefully using the COMMIT_ON_RETURN column of the SYSIBM.SYSPROCEDURES catalog table to indicate that DB2 should issue an implicit COMMIT on behalf of the stored procedure upon return from the CALL statement. This can reduce the length of time locks are held and can reduce network traffic, especially when the request is from an application using DRDA level 1. When you use COMMIT_ON_RETURN, any updates made by the client before calling the stored procedure are committed with the stored procedure changes. See Section 6 of *Application Programming and SQL Guide* for more information.
- For applications using private protocol, you can achieve the best performance for distributed applications if every read-only cursor uses its own conversation. Resource constraints could conflict with making multiple conversations available to an application. For more information, see *Installation Guide* .

The Serving System: This is the system on which the data resides. For access using DB2 private protocol, this is the DB2 system on which the SQL is dynamically executed. For access using DRDA, this 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 5-10. Using EXPLAIN to Improve SQL Performance” on page 5-261 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.) Your request could be queued while waiting for an available thread.
- The dispatching priority of database access threads on the remote system. A low dispatching priority could impede your application's distributed performance. See “Prioritize Resources” on page 5-74 for more information.
- For instructions on avoiding RACF calls at the server, see “Controlling Requests from Remote Applications” on page 3-71, and more particularly “Do You Manage Inbound IDs Through DB2 or RACF?” on page 3-76.

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 problems.

The Logical and Physical Connection Between the Requester and Server: DB2 can use two types of logical connections between requesters and servers:

- System Network Architecture LU6.2 DB2 uses APPC/VTAM to implement LU6.2 architecture.

  Using ACF/VTAM Version 4 Release 4 or later releases reduces the processing costs of VTAM overhead associated with DRDA conversations. The reductions benefit both requesters and servers but is not shown in the accounting data for the end user.
- Transmission Control Protocol/Internet Protocol (TCP/IP)

For information on tuning these connections, see Section 3 of *Installation Guide* .

How Block Fetch Improves Performance

DB2 has an important capability called *block fetch* that can significantly affect the number of messages sent across the network. Used with cursors that will not update data, 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 employs two different types of block fetch:

- *Limited block fetch* is used for both access using DRDA or DB2 private protocol.
- *Continuous block fetch* is only used by DB2 private protocol.

In terms of response times, the continuous block method is more efficient 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 resources in the form of conversations. Switching from continuous block to limited block allows applications to run when resources (conversations) are critical.

Both forms of blocking can be in use at the same time by the same agent.

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.

Continuous Block Fetch (Private Protocol Only): Continuous block fetch is a DB2-defined method that uses one conversation for each open cursor. When using this method, a single request message is sent from the requester to the server. The server then fills a buffer with response information and transmits the buffer back to the requester. Processing at the requester is asynchronous with processing at the server. The server continues to fill buffers and transmit them to the requester until suspended by VTAM because of the pacing limits defined for the conversation. The continuous block method, then, only requires a single request message.

To understand what happens, consider the sequence of events in an application that both reads and updates data at a remote location:

1. The local subsystem, A, sends a message to open a cursor and begin a fetch at the remote subsystem, B.
2. A receives the row from B and processes it.
3. A sends a message to update the row at B.
4. A repeats steps 2 and 3 for the second and later rows.

But if the application does not need to update subsystem B, then step 3 can be eliminated. Using continuous block fetch, the sequence would go like this:

1. A sends a message to open a cursor and begin a fetch 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.

Updatable cursors cannot use either type of block fetch, which can retrieve the data before the application actually asks for it. Updates using cursors require that the application be synchronized with the cursor position at the responding location.

General-use Programming Interface

Using FOR FETCH ONLY to Ensure Block Fetch

You can ensure block fetching by including the clause FOR FETCH ONLY in your DECLARE CURSOR statement, as in the following example:

```
EXEC SQL
  DECLARE THISEMP CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
    FROM DSN8510.EMP
    WHERE WORKDEPT = 'D11'
    FOR FETCH ONLY
END-EXEC.
```

If you use FOR FETCH ONLY, you cannot also use FOR UPDATE OF.

In general, DB2 can use block fetch for a query if:

- The declare cursor statement includes a SELECT that is specified to be FOR FETCH ONLY.
- The result table of the cursor is read-only. This applies to static and dynamic cursors except for read-only views. See Chapter 6 of *SQL Reference* for more information about declaring a cursor as read-only.
- FOR FETCH ONLY was not specified and the result table of the cursor is not read-only, and no static DELETE WHERE CURRENT OF or UPDATE WHERE CURRENT OF is in the query, and there are no dynamic statements in the program.

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.

End of General-use Programming Interface

General-use Programming Interface

Using CURRENTDATA(NO) to Ensure Block Fetch

If any of the following are true, the cursors are read-only and NOT ambiguous, and block fetch is automatically used for the query:

- The cursor is based on a read-only SELECT statement.
- The cursor is declared with a DISTINCT, ORDER BY, or GROUP BY clause.
- The cursor is declared FOR FETCH ONLY.

A cursor is considered updatable if:

- The cursor is declared with a FOR UPDATE clause.
- A DELETE WHERE CURRENT statement is associated with the cursor.

If none of the above are true, and a PREPARE or EXECUTE IMMEDIATE statement appears in the same program, the cursor is considered ambiguous.

When you specify CURRENTDATA(NO) on either the BIND PACKAGE or BIND PLAN commands, block fetching is allowed for ambiguous cursors. The default is CURRENTDATA(YES); therefore, to take advantage of block fetching for ambiguous cursors, you must be sure to specify CURRENTDATA(NO).

Table 92 on page 5-321 summarizes the effects of CURRENTDATA and cursor type on block fetch.

Table 92. Effect of CURRENTDATA and Cursor Type on Block Fetch

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

End of General-use Programming Interface

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.

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 *Command Reference*. See also:

"Monitoring Threads" on page 4-37
"The Command DISPLAY LOCATION" on page 4-63

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, we recommend that you turn on the statistics class 4 trace and always keep it active. This statistics trace covers error situations surrounding indoubt 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 151 on page 5-322 shows the relationship of the accounting class 1 and 2 times and the requester and server accounting records. Figure 152 on page 5-324 and Figure 153 on page 5-324 show the server and requester distributed data facility blocks from the DB2 PM accounting long trace.

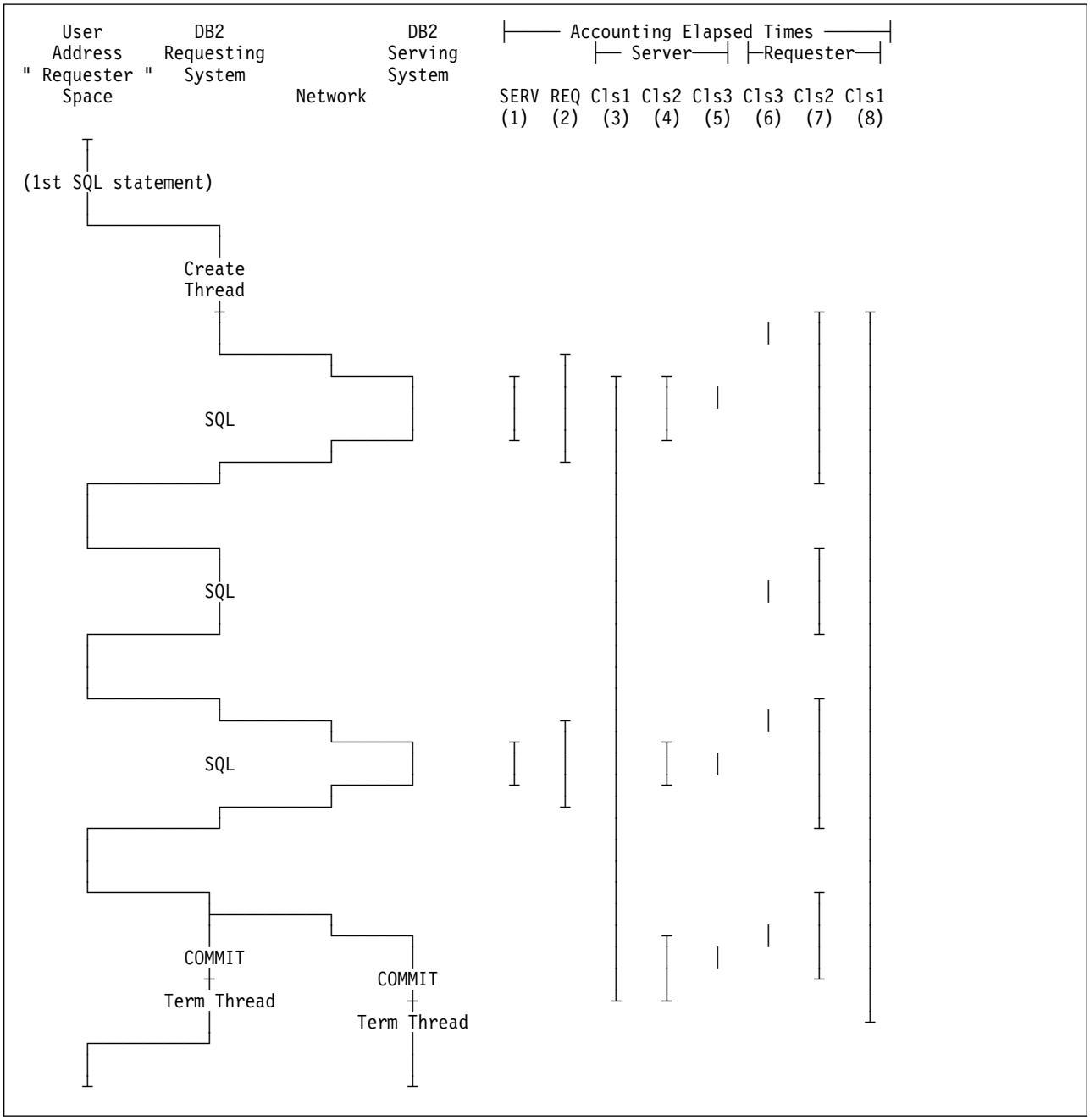


Figure 151. Elapsed Times in a DDF Environment as Reported by DB2 PM. These times are valid for access that uses either DRDA or private protocol (except as noted).

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) - SERV

  This time is reported in the SERVER ELAPSED TIME (**A**) field in the DDF block of the DB2 PM accounting long trace. It represents the amount of elapsed time spent at the server between the receipt of the SQL statement until the answer is sent to VTAM. (This does not apply to access that uses DRDA.)

- (2) - REQ

  This time is reported in the REQUESTER ELAP.TIME (B) field in the DDF block of the DB2 PM accounting long trace. It represents the amount of elapsed time spent at the requester between the sending of the SQL statement and the receipt of the answer from the server.

- (3) - 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 system. It represents the elapsed time from the creation of the database access thread until the termination of the database access thread.

- (4) - 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 statement and the commit at the server.

- (5) - 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 for the serving DB2 subsystem. It represents the amount of time the serving DB2 system spent suspended waiting for locks or I/O.

- (6) - 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.

- (7) - 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 system. It represents the elapsed time from when the application passed the SQL statement to the local DB2 system until return. This is considered “In DB2” time.

- (8) - 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 system. It represents the elapsed time from the creation of the allied distributed thread until the termination of the allied distributed thread.

Figure 151 on page 5-322 highlights that the Class 2 "in DB2" elapsed time at the requester includes the time that elapses while the requester is waiting for a response from the server. To remove that time, subtract the requester elapsed time (see the REQ column in Figure 151) from the Class 2 elapsed time.

However, 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             : VTAMA          CONVERSATIONS INITIATED: D  1  TRANSACT.SENT:   1  MESSAGES SENT    :              2
PRODUCT ID         : DB2            #CONVERSATIONS QUEUED  : C  0  #COMMT(1)SENT:   1  MESSAGES RECEIVED:              2
PRODUCT VERSION    : V4 R1 M0       SUCCESSFULLY ALLOC.CONV: E N/A #ROLLB(1)SENT:   0  BYTES SENT       :            736
METHOD             : APPL DIRECTED  CONVERSATION TERMINATED:   N/A SQL SENT:        1  BYTES RECEIVED   :            292
REQUESTER ELAP.TIME: B    0.119155  MAX OPEN CONVERSATIONS :   N/A ROWS RECEIVED:   1  BLOCKS RECEIVED  :              0
 A                                   F                             H                    G
SERVER ELAPSED TIME:           N/A  #CONT->LIM.BL.FTCH SWCH:   N/A MSG.IN BUFFER:   0  STMT BOUND AT SER:              0
SERVER CPU TIME    :           N/A  #DDF ACCESSES          :    1
#COMMIT(2) SENT    :             0  #COMMIT(2) RESP.RECV.  :    0  #PREPARE SENT:   0  #FORGET RECEIVED :              0
#BACKOUT(2) SENT   :             0  #BACKOUT(2) RESP.RECV. :    0  #LASTAGN.SENT:   0
```

Figure 152. DDF Block of a Requester Thread from a DB2 PM Accounting Long Trace

```
---- DISTRIBUTED ACTIVITY ------------------------------------------------------------------------------------------
REQUESTER          : VTAMB            TRANSACTIONS RECV. :    1.00 MESSAGES SENT    :      2.00  MSG.IN BUFFER :       0.00
PRODUCT ID         : DB2              #COMMIT(1) RECEIVED:       1 MESSAGES RECEIVED:      2.00  ROWS SENT     :       1.00
PRODUCT VERSION    : V4 R1 M0         #ROLLBK(1) RECEIVED:       0 BYTES SENT       :    292.00  BLOCKS SENT   :       0.00
METHOD             : APPL DIRECTED    SQL RECEIVED       :    1.00 BYTES RECEIVED   :    736.00  CONV.INITIATED:       1.00
#COMMIT(2) RECEIVED:              0   #COMMIT(2) RES.SENT:       0 #PREPARE RECEIVED:         0  #DDF ACCESSES :          1
#BCKOUT(2) RECEIVED:              0   #BACKOUT(2)RES.SENT:       0 #LAST AGENT RECV.:         0  #FORGET SENT  :          0
#COMMIT(2) PERFORM.:              1   #BACKOUT(2)PERFORM.:       0 #THREADS INDOUBT :         0
```

Figure 153. 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 VTAM conversations is reported in several fields:

- The number of conversation requests queued during allocation is reported as CONVERSATIONS QUEUED (**C**).
- The number of conversation allocations is reported as CONVERSATIONS INITIATED (**D**).
- The number of successful conversation allocations is reported as SUCCESSFULLY ALLOC.CONV (**E**).
- The number of times a switch was made from continuous block fetch to limited block fetch is reported as CONT->LIM.BL.FTCH (**F**). 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.

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 (**G**).

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 (**H**). 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.

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.

Using DB2 PM Accounting Reports to Monitor Distributed Processing

The DB2 PM accounting report, short layout, can be used to monitor the distributed activity between your DB2 subsystem and other database management systems in the network.

The accounting report, short layout, is often ordered by PRIMAUTH/PLANNAME as the default. With distributed activity, ordering by PRIMAUTH/PLANNAME/REQLOC provides additional separation of data according to the requesting locations. Additionally, it provides separation of server thread accounting records from non-distributed accounting records and requester accounting records.

Accounting records that have a SERVER location block are created by any activity involving one or more allied distributed threads. This type of thread executes on the local (requesting) DB2, which is requesting data from a remote (serving) DB2 location. Work done at the serving location is performed by a database access thread (DBAT).

Accounting records that have a REQUESTER location block are created when processing activity involves one or more database access threads. This type of thread executes on the local (serving) DB2 and is created in response to a request for data from a remote (requesting) DB2 location.

| With access using DRDA, it is possible that a database access thread subsequently
| makes a request of another DB2 by using private protocol. In this case, DB2 PM
| calls that thread a DBAT distributed thread, and the accounting record contains
| both a requester location block for the DRDA access activity and a server location
| block for the private protocol access activity.

Merged Accounting Trace

With DB2 PM, you can create merged accounting traces based on job input from both the requester and server subsystem accounting records. For multi-site update, trace records from more than one remote site can be included.

Using RMF to Monitor Distributed Processing

If you use RMF to monitor DDF work, it's important to understand how DDF is using the enclave SRBs described in “Using Workload Manager to Set Performance Objectives” on page 5-124. 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

“Comparing Active and Inactive Threads” on page 5-122 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 154 on page 5-327 contrasts the two types of threads and how they are managed by SRM.

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 conributes 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.

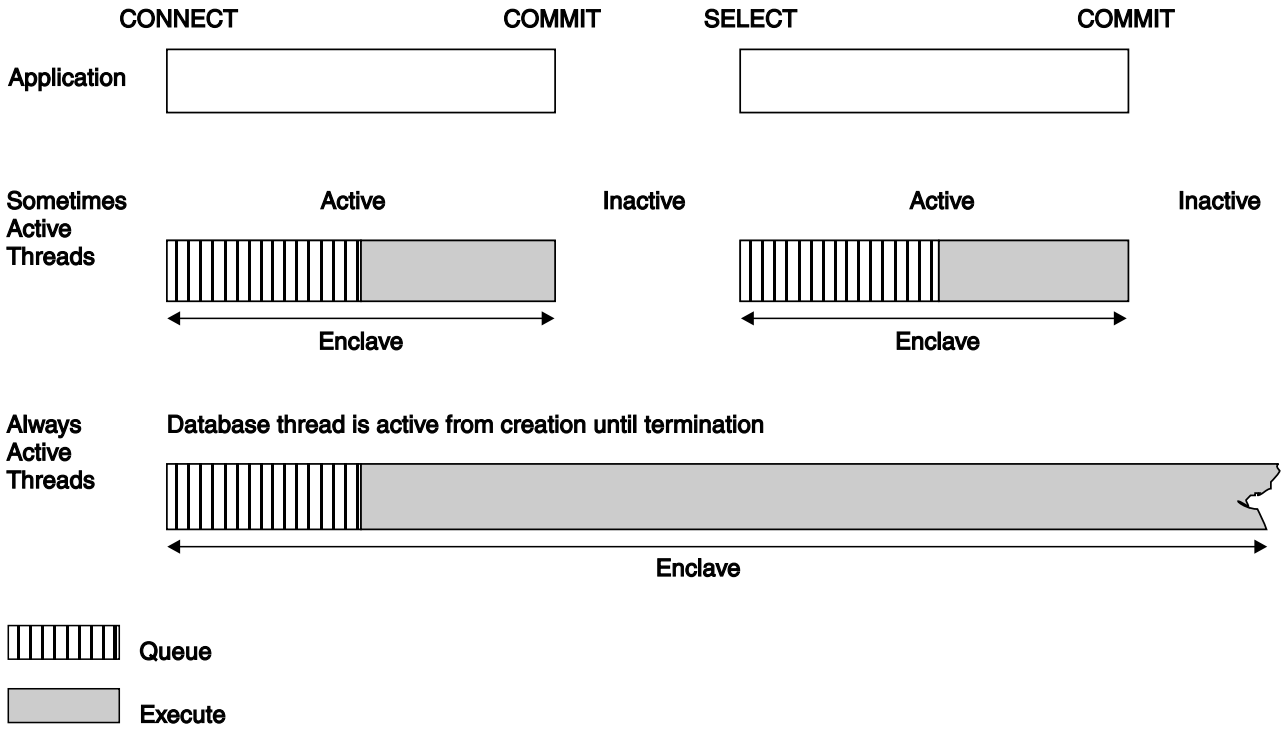


Figure 154. Contrasting 'Always Active' vs. 'Sometimes Active' Threads

Monitoring and Tuning Stored Procedures

| Table 93 on page 5-328 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.

Table 93. Comparing WLM-established and DB2-established Stored Procedures

DB2-established	WLM-established	More Information
There is a single stored procedure address space: • A failure in one stored procedure can affect other stored procedures that are running in that address space. • Because of storage that language products need below the 16MB line, it can be difficult to support more than 50 stored procedures running at the same time.	There can be many stored procedures address spaces: • It is possible to isolate stored procedures from one another so that failures do not affect other stored procedures. • Reduces demand for storage below the 16MB line and thereby removes the limitation on the number of stored procedures that can run concurrently.	"Controlling Address Space Storage" on page 5-329
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 5-124
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.	"Using Workload Manager to Set Performance Objectives" on page 5-124
No ability to customize the environment.	Each stored procedures address space is associated with a WLM *environment* that you specify. An environment is an attribute associated with one or more stored procedures. The environment determines which JCL procedure is used to run a particular stored procedure.	"Assigning Stored Procedures to WLM Application Environments" on page 5-329
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.	Section 6 of *Application Programming and SQL Guide*
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.	Section 6 of *Application Programming and SQL Guide*
Stored procedures access protected MVS resources with the authority of the stored procedures address space.	Stored procedures can access protected MVS resources with the SQL user's RACF authority.	Section 3 of *Administration Guide*

Controlling Address Space Storage

To maximize the number of stored procedures that can run concurrently in a stored procedures address space, use the following guidelines:

- Set REGION size for the stored procedures 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:
  - Link editing programs above the line with AMODE(31) and RMODE(ANY) attributes
  - Using the RES and DATA(31) compiler options for COBOL programs
- Limiting storage required by IBM Language Environment for MVS & VM 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 100KB below the 16MB line. Each TCB that runs in a WLM-established stored procedures address space uses approximately 200KB.

DB2 needs extra storage for stored procedures in the WLM-established address space because you can create both main and sub programs, and DB2 must create an environment for each.

You must have Language Environment to run stored procedures. Your requirements can differ significantly depending on your release of Language Environment.

Dynamically Extending Load Libraries: We recommend using 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 Stored Procedures to WLM Application Environments

Workload manager routes work to stored procedure address spaces based on the environment name and service class associated with the stored procedure. You must use WLM panels to associate an application environment name with the JCL procedure used to start an address space. See *MVS/ESA Planning: Workload Management* for details about workload management panels.

There are other tasks that must be completed before a stored procedure 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 stored procedures will not hang for an unlimited amount of time.
2. If you want to migrate any 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 environment or service class is not used for a long time, WLM creates on demand at least one address space for each combination of WLM environment name and service class that is encountered in the workload. For example, if there are five environment names that each have six possible service classes, and all those combinations are in demand, it is possible to have 30 stored procedure address spaces.

   To prevent creating too many address spaces, create a relatively small number of WLM environments and MVS service classes.
3. Use the WLM application environment panels to associate the environment name with the JCL procedure. Figure 155 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 . . . . . . . . . WLMENV2
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 155. 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. Update the WLM_ENV column of SYSIBM.SYSPROCEDURES to associate a stored procedure with an application environment.

   ```
   UPDATE SYSIBM.SYSPROCEDURES
     SET WLM_ENV='WLM_ENV2'
     WHERE PROCEDURE='BIGPROC';
   ```

5. Using the WLM install utility, 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 allows those procedures to pick up the application environment from the WLM_ENV column of SYSIBM.SYSPROCEDURES.
8. Begin running stored procedures.

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 156.

```
PRIMAUTH: USRT001  PLANNAME: IT8EC

AVERAGE       APPL (CLASS 1)   DB2 (CLASS 2)   IFI (CLASS 5)    CLASS 3 SUSP.   AVERAGE TIME  AV.EVENT
------------  --------------  --------------  --------------    --------------  ------------  --------
ELAPSED TIME        0.123676        0.053400             N/P    LOCK/LATCH           0.000000      0.00
CPU TIME            0.012648        0.009332             N/P    SYNCHRON. I/O        0.040742      1.00
 TCB                0.004097        0.001719             N/P    OTHER READ I/O       0.000000      0.00
 TCB-STPROC  A      0.008551        0.007613             N/A    OTHER WRTE I/O       0.000000      0.00
 CPU-PARALL         0.000000        0.000000             N/A    SER.TASK SWTCH       0.000000      0.00
SUSPEND TIME             N/A        0.040742             N/A    ARC.LOG(QUIES)       0.000000      0.00
 TCB                     N/A        0.040742             N/A    ARC.LOG READ         0.000000      0.00
 CPU-PARALL              N/A        0.000000             N/A    DRAIN LOCK           0.000000      0.00
NOT ACCOUNT.             N/A        0.003327             N/A    CLAIM RELEASE        0.000000      0.00
DB2 ENT/EXIT             N/A            8.00             N/A    PAGE LATCH           0.000000      0.00
EN/EX-STPROC             N/A           36.00             N/A    STORED PROC.  B      0.000000      0.00
DCAPT.DESCR.             N/A             N/A             N/P    NOTIFY MSGS.         0.000000      0.00
LOG EXTRACT.             N/A             N/A             N/P    GLOBAL CONT.         0.000000      0.00
NOT NULL                   1               1               0    TOTAL CLASS 3        0.040742      1.00

⋮

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 156. Partial Long Accounting Report, Server - Stored Procedures

Descriptions of Fields:

- The number of calls to stored procedures is indicated in **C**.
- 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 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 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.

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 be rejected.

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 xi.

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 job DSNTEJ1. The CREATE INDEX statements for the sample tables are not shown here; they, too, are created by the DSNTEJ1 sample job.

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 DSN8510.PROJ. For convenience in interpreting the examples, the department and employee tables are listed here in full.

Activity Table (DSN8510.ACT)

The activity table describes the activities that can be performed during a project. The table resides in database DSN8D51A and is created with:

```
CREATE TABLE DSN8510.ACT
      (ACTNO    SMALLINT       NOT NULL,
       ACTKWD   CHAR(6)        NOT NULL,
       ACTDESC  VARCHAR(20)    NOT NULL,
       PRIMARY KEY (ACTNO)             )
  IN DSN8D41A.DSN8S41P:
```

Content

Table 94 shows the content of the columns.

Table 94. 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 95. Indexes of the Activity Table

Name	On Column	Type of Index
DSN8510.XACT1	ACTNO	Primary, ascending
DSN8510.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 (DSN8510.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 98 on page X-9, resides in table space DSN8D51A.DSN8S51D and is created with:

```
CREATE TABLE DSN8510.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 DSN8D51A.DSN8S51D;
```

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 DSN8510.DEPT
      FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8510.DEPT
              ON DELETE CASCADE;

ALTER TABLE DSN8510.DEPT
      FOREIGN KEY RDE (MGRNO) REFERENCES DSN8510.EMP
              ON DELETE SET NULL;
```

Content

Table 96 shows the content of the columns.

Table 96. 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 97. Indexes of the Department Table

Name	On Column	Type of Index
DSN8510.XDEPT1	DEPTNO	Primary, ascending
DSN8510.XDEPT2	MGRNO	Ascending
DSN8510.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 98. DSN8510.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 (DSN8510.EMP)

The employee table identifies all employees by an employee number and lists basic personnel information.

The table shown in Table 101 on page X-12 and Table 102 on page X-13 resides in the partitioned table space DSN8D51A.DSN8S51E. 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 DSN8510.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 DSN8510.DEPT
                 ON DELETE SET NULL                       )
  EDITPROC  DSN8EAE1
  IN DSN8D51A.DSN8S51E;
```

Content

Table 99 on page X-11 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 99. 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 100. Indexes of the Employee Table

Name	On Column	Type of Index
DSN8510.XEMP1	EMPNO	Primary, partitioned, ascending
DSN8510.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 101. Left Half of DSN8510.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 102. Right Half of DSN8510.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

Project Table (DSN8510.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 DSN8D51A. 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 DSN8410.PROJ
               (PROJNO   CHAR(6) PRIMARY KEY NOT NULL,
                PROJNAME VARCHAR(24)    NOT NULL WITH DEFAULT
                  'PROJECT NAME UNDEFINED',
                DEPTNO   CHAR(3)        NOT NULL REFERENCES
                  DSN8410.DEPT ON DELETE RESTRICT,
                RESPEMP  CHAR(6)        NOT NULL REFERENCES
                  DSN8410.EMP ON DELETE RESTRICT,
                PRSTAFF  DECIMAL(5, 2)          ,
                PRSTDATE DATE                   ,
                PRENDATE DATE                   ,
                MAJPROJ  CHAR(6))
        IN DSN8D41A.DSN8S41P;
```

Because the table is self-referencing, the foreign key for that restraint must be added later with:

```
ALTER TABLE DSN8510.PROJ
      FOREIGN KEY RPP (MAJPROJ) REFERENCES DSN8510.PROJ
              ON DELETE CASCADE;
```

Content

Table 103 shows the content of the columns.

Table 103. 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 104. Indexes of the Project Table

Name	On Column	Type of Index
DSN8510.XPROJ1	PROJNO	Primary, ascending
DSN8510.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 (DSN8510.PROJACT)

The project activity table lists the activities performed for each project. The table resides in database DSN8D51A. 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 DSN8510.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 DSN8510.PROJ
                                      ON DELETE RESTRICT,
       FOREIGN KEY RPAA (ACTNO) REFERENCES DSN8510.ACT
                                      ON DELETE RESTRICT)
  IN DSN8D41A.DSN8S41P;
```

Content

Table 105 shows the content of the columns.

Table 105. 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 106. Index of the Project Activity Table

Name	On Columns	Type of Index
DSN8510.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 (DSN8510.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 DSN8D51A. 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 DSN8510.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 DSN8510.PROJACT
                                       ON DELETE RESTRICT,
       FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8510.EMP
                                       ON DELETE RESTRICT)
  IN DSN8D41A.DSN8S41P;
```

Content

Table 107 shows the content of the columns.

Table 107. 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 108. Indexes of the Employee to Project Activity Table

Name	On Columns	Type of Index
DSN8510.XEMPPROJACT1	PROJNO, ACTNO, EMSTDATE, EMPNO	Unique, ascending
DSN8510.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 157 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.

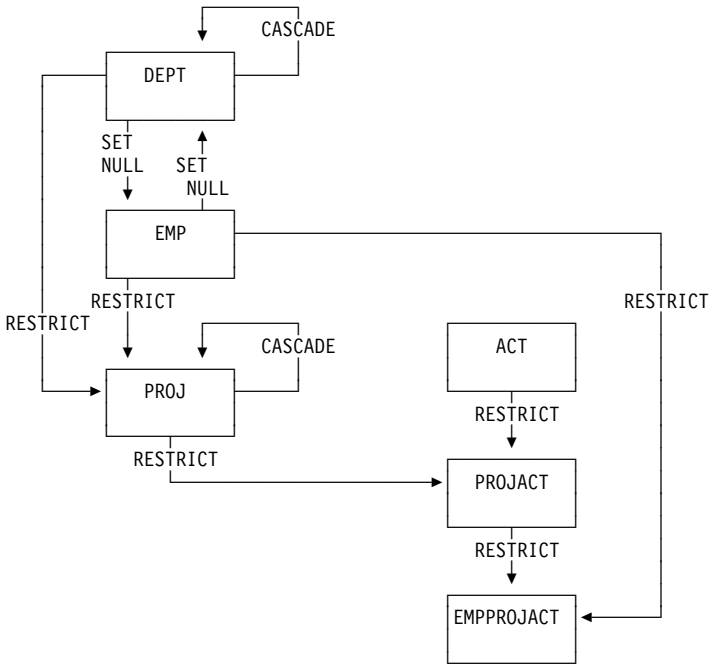


Figure 157. Relationships among Tables in the Sample Application. Arrows point from parent tables to dependent tables.

Views on the Sample Tables

DB2 creates a number of views on the sample tables for use in the sample applications. Table 109 indicates the tables on which each view is defined and the sample applications that use the view. All view names have the qualifier DSN8510.

Table 109. Views on sample tables

View name	On tables or views	Used in application
VDEPT	DEPT	Organization Project
VHDEPT	DEPT	Distributed organization
VEMP	EMP	Distributed organization Organization Project
VPROJ	PROJ	Project
VACT	ACT	Project
VEMPPROJACT	EMPROJACT	Project
VDEPMG1	DEPT EMP	Organization
VEMPDPT1	DEPT EMP	Organization
VASTRDE1	DEPT	
VASTRDE2	VDEPMG1 EMP	Organization
VPROJRE1	PROJ EMP	Project
VPSTRDE1	VPROJRE1 VPROJRE2	Project
VPSTRDE2	VPROJRE1	Project
VSTAFAC1	PROJACT ACT	Project
VSTAFAC2	EMPPROJACT ACT EMP	Project
VPHONE	EMP DEPT	Phone
VEMPLP	EMP	Phone

The SQL statements that create the sample views are shown below.

```
CREATE VIEW DSN8510.VDEPT
   AS SELECT ALL      DEPTNO  ,
                      DEPTNAME,
                      MGRNO   ,
                      ADMRDEPT
   FROM DSN8510.DEPT;
```

```
#                      CREATE VIEW DSN8510.VHDEPT
#                         AS SELECT ALL      DEPTNO  ,
#                                            DEPTNAME,
#                                            MGRNO   ,
#                                            ADMRDEPT,
#                                            LOCATION
#                         FROM DSN8510.DEPT;

#                      CREATE VIEW DSN8510.VEMP
#                         AS SELECT ALL      EMPNO   ,
#                                            FIRSTNME,
#                                            MIDINIT ,
#                                            LASTNAME,
#                                            WORKDEPT
#                         FROM DSN8510.EMP;

#                      CREATE VIEW DSN8510.VPROJ
#                         AS SELECT ALL
#                                   PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,
#                                   PRSTDATE, PRENDATE, MAJPROJ
#                         FROM DSN8510.PROJ ;

#                      CREATE VIEW DSN8510.VACT
#                            AS SELECT ALL   ACTNO   ,
#                                            ACTKWD  ,
#                                            ACTDESC
#                                 FROM DSN8510.ACT ;

#                      CREATE VIEW DSN8510.VPROJACT
#                            AS SELECT ALL
#                                  PROJNO,ACTNO, ACSTAFF, ACSTDATE, ACENDATE
#                                  FROM DSN8510.PROJACT ;

#                      CREATE VIEW DSN8510.VEMPPROJACT
#                            AS SELECT ALL
#                                EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE
#                                FROM DSN8510.EMPPROJACT ;

#                      CREATE VIEW DSN8510.VDEPMG1
#                            (DEPTNO, DEPTNAME, MGRNO, FIRSTNME, MIDINIT, LASTNAME, ADMRDEPT)
#                          AS SELECT ALL
#                             DEPTNO, DEPTNAME, EMPNO, FIRSTNME, MIDINIT, LASTNAME, ADMRDEPT
#                              FROM DSN8510.DEPT LEFT OUTER JOIN DSN8510.EMP
#                              ON MGRNO = EMPNO ;

#                      CREATE VIEW DSN8510.VEMPDPT1
#                            (DEPTNO, DEPTNAME, EMPNO, FRSTINIT, MIDINIT,
#                             LASTNAME, WORKDEPT)
#                          AS SELECT ALL
#                             DEPTNO, DEPTNAME, EMPNO, SUBSTR(FIRSTNME, 1, 1), MIDINIT,
#                             LASTNAME, WORKDEPT
#                              FROM DSN8510.DEPT RIGHT OUTER JOIN DSN8510.EMP
#                              ON WORKDEPT = DEPTNO ;
```

```
#                     CREATE VIEW DSN8510.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 DSN8510.VDEPMG1 D1, DSN8510.VDEPMG1 D2
#                             WHERE D1.DEPTNO = D2.ADMRDEPT ;
#
#                     CREATE VIEW DSN8510.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 DSN8510.VDEPMG1 D1, DSN8510.EMP E2
#                              WHERE D1.DEPTNO = E2.WORKDEPT;
#
#                     CREATE VIEW DSN8510.VPROJRE1
#                       (PROJNO,PROJNAME,PROJDEP,RESPEMP,FIRSTNME,MIDINIT,LASTNAME,MAJPROJ)
#                        AS SELECT ALL
#                           PROJNO,PROJNAME,DEPTNO,EMPNO,FIRSTNME,MIDINIT,LASTNAME,MAJPROJ
#                         FROM DSN8510.PROJ, DSN8510.EMP
#                         WHERE RESPEMP = EMPNO ;
#
#                     CREATE VIEW DSN8510.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 DSN8510.VPROJRE1 P1,
#                            DSN8510.VPROJRE1 P2
#                          WHERE P1.PROJNO = P2.MAJPROJ ;
#
#                     CREATE VIEW DSN8510.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 DSN8510.VPROJRE1 P1
#                            WHERE NOT EXISTS
#                              (SELECT * FROM DSN8510.VPROJRE1 P2
#                                WHERE P1.PROJNO = P2.MAJPROJ) ;
```

```
#                     CREATE VIEW DSN8510.VFORPLA
#                       (PROJNO,PROJNAME,RESPEMP,PROJDEP,FRSTINIT,MIDINIT,LASTNAME)
#                        AS SELECT ALL
#                           F1.PROJNO,PROJNAME,RESPEMP,PROJDEP, SUBSTR(FIRSTNME, 1, 1),
#                           MIDINIT, LASTNAME
#                          FROM DSN8510.VPROJRE1 F1 LEFT OUTER JOIN DSN8510.EMPPROJACT F2
#                          ON F1.PROJNO = F2.PROJNO;

#                     CREATE VIEW DSN8510.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 DSN8510.PROJACT PA, DSN8510.ACT AC
#                          WHERE PA.ACTNO = AC.ACTNO ;

#                     CREATE VIEW DSN8510.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 DSN8510.EMPPROJACT EP, DSN8510.ACT AC, DSN8510.EMP EM
#                          WHERE EP.ACTNO = AC.ACTNO  AND EP.EMPNO = EM.EMPNO ;

#                     CREATE VIEW DSN8510.VPHONE
#                                  (LASTNAME,
#                                   FIRSTNAME,
#                                   MIDDLEINITIAL,
#                                   PHONENUMBER,
#                                   EMPLOYEENUMBER,
#                                   DEPTNUMBER,
#                                   DEPTNAME)
#                        AS SELECT ALL      LASTNAME,
#                                           FIRSTNME,
#                                           MIDINIT ,
#                                           VALUE(PHONENO,'    '),
#                                           EMPNO,
#                                           DEPTNO,
#                                           DEPTNAME
#                        FROM DSN8510.EMP, DSN8510.DEPT
#                        WHERE WORKDEPT = DEPTNO;

#                     CREATE VIEW DSN8510.VEMPLP
#                                  (EMPLOYEENUMBER,
#                                   PHONENUMBER)
#                        AS SELECT ALL      EMPNO   ,
#                                           PHONENO
#                        FROM DSN8510.EMP ;
```

Storage of Sample Application Tables

Figure 158 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.

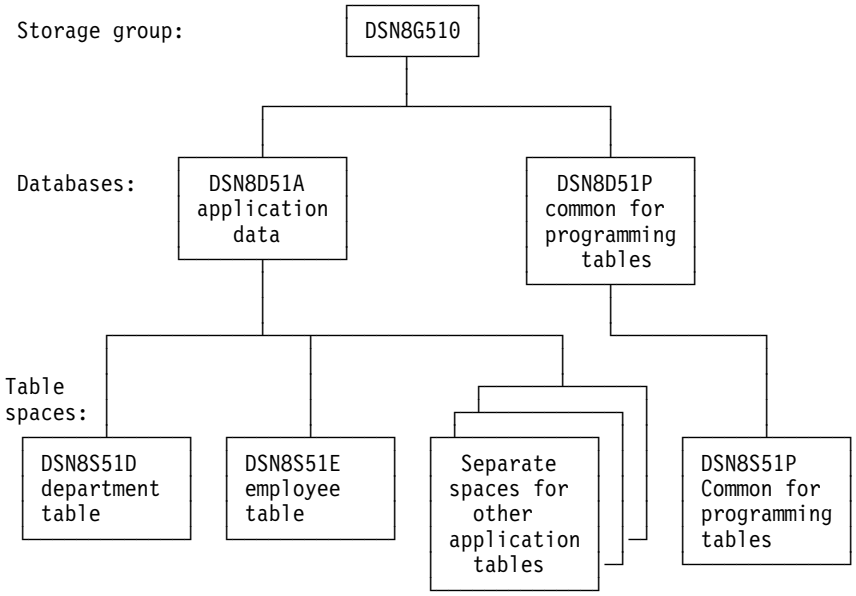


Figure 158. Relationship among Sample Databases and Table Spaces

In addition to the storage group and databases shown in Figure 158, the storage group DSN8G51U and database DSN8D51U 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 DSN8G510
  VOLUMES (DSNV01)
  VCAT  DSNC510
  PASSWORD DSNDEFPW;
```

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 DSN8D51A
  STOGROUP DSN8G510
  BUFFERPOOL BP0;

CREATE DATABASE DSN8D51P
  STOGROUP DSN8G510
  BUFFERPOOL BP0;
```

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 DSN8D51A database, using the default space attributes.

```
CREATE TABLESPACE DSN8S51D
  IN DSN8D51A
  USING STOGROUP DSN8G510
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  DSETPASS DSN8;

CREATE TABLESPACE DSN8S51E
  IN DSN8D51A
  USING STOGROUP DSN8G510
        PRIQTY 20
        SECQTY 20
        ERASE NO
  NUMPARTS 4
    (PART 1 USING STOGROUP DSN8G510
                  PRIQTY 12
                  SECQTY 12,
     PART 3 USING STOGROUP DSN8G510
                  PRIQTY 12
                  SECQTY 12)
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  DSETPASS DSN8
  COMPRESS YES;

CREATE TABLESPACE DSN8S51C
  IN DSN8D51P
  USING STOGROUP DSN8G510
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO
  DSETPASS DSN8;

CREATE TABLESPACE DSN8S41P
  IN DSN8D51A
  USING STOGROUP DSN8G510
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE ROW
  BUFFERPOOL BP0
  CLOSE NO
  DSETPASS DSN8;
```

```
CREATE TABLESPACE DSN8S51R
  IN DSN8D51A
  USING STOGROUP DSN8G510
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  DSETPASS DSN8;

CREATE TABLESPACE DSN8S41S
   IN DSN8D41A
   USING STOGROUP DSN8G410
             PRIQTY 20
             SECQTY 20
             ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   DSETPASS DSN8;
```

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 xi.

DB2 provides installation-wide exit points to routines that you provide. They are described under the following headings:

Connection and Sign-On Routines

Your DB2 system 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.

For a general view of the roles of the exit routines in assigning authorization IDs, see “Chapter 3-4. Controlling Access to a DB2 Subsystem” on page 3-63. 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 X-74 applies to these routines, but with the following exception to the description of execution environments:

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. But, 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 3-71.)

- 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 3-64. For instructions on controlling the IDs associated with sign-on requests, see "Processing Sign-ons" on page 3-68.

EXPL for Connection and Sign-on Routines

Figure 159 shows how the parameter list points to other information.

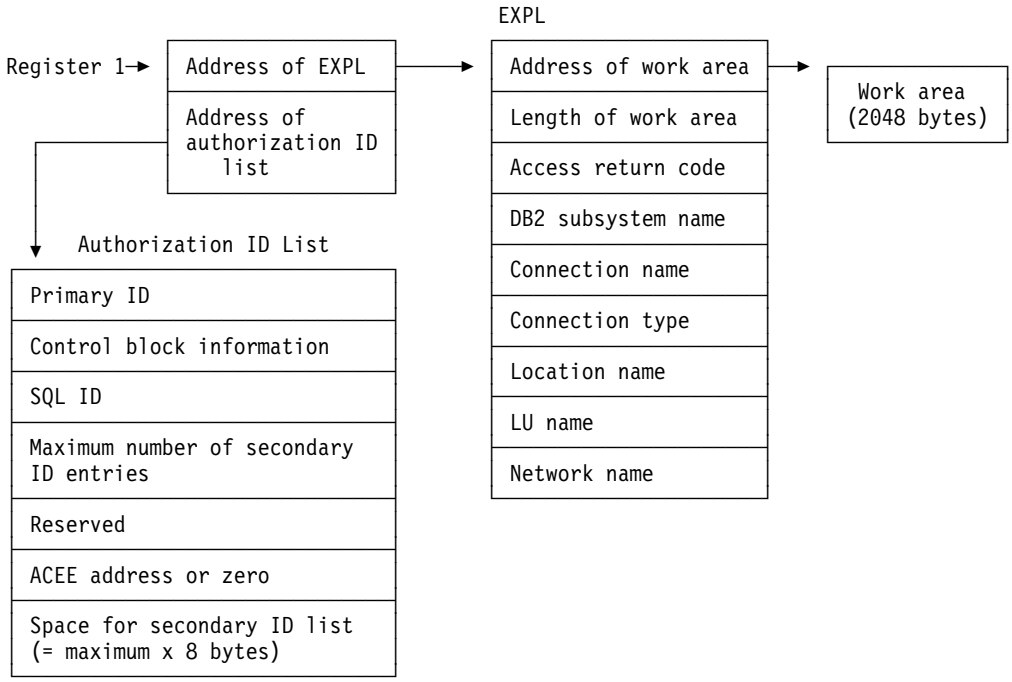


Figure 159. How a Connection or Sign-on Parameter List Points to Other Information

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 110. 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
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: 0 Access allowed; DB2 continues processing. 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 111 on page X-29. The description is given by macro DSNDAIDL.

Table 111. 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.
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 X-31.

   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 3-68 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
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.

General Considerations

The routine executes in the *ssnm*DBM1 address space of DB2.

"General Considerations for Writing Exit Routines" on page X-74 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 112 on page X-37).
- 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 is started up this exit is taken to allow the external authorization checking application to perform any required setup prior to authorization checking, like loading authorization profiles into storage, and so on.

- 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.

- Changing IDs by using SET CURRENT SQLID

People often use the SET CURRENT SQLID statement to change the ID that is used to create objects. The CURRENT SQLID can be the primary ID, one of the secondary IDs, or, if SYSADM is running the job, any character string.

The exit routine receives the ACEE of the primary authorization ID (field XAPLUPRM), and it also receives the value of the CURRENT SQLID (in field XAPLUCHK). DB2 performs the authorization on the value in field XAPLUCHK.

- Invalidating plans or 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 a privilege that the plan or package depends on is revoked, and if you want to invalidate the plan or package, you must use the SQL GRANT statement to grant the revoked privilege and then use the SQL statement REVOKE 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 plans and packages

  The results of authorization checks on EXECUTE authority are not cached when those checks are performed by the exit routine.

- 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

  The results of authorization checks on the EXECUTE privilege for packages are cached (assuming that package 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 you must use the SQL statements GRANT and REVOKE to refresh the cache.

Parameter List for the Access Control Authorization Routine

Figure 160 shows how the parameter list points to other information.

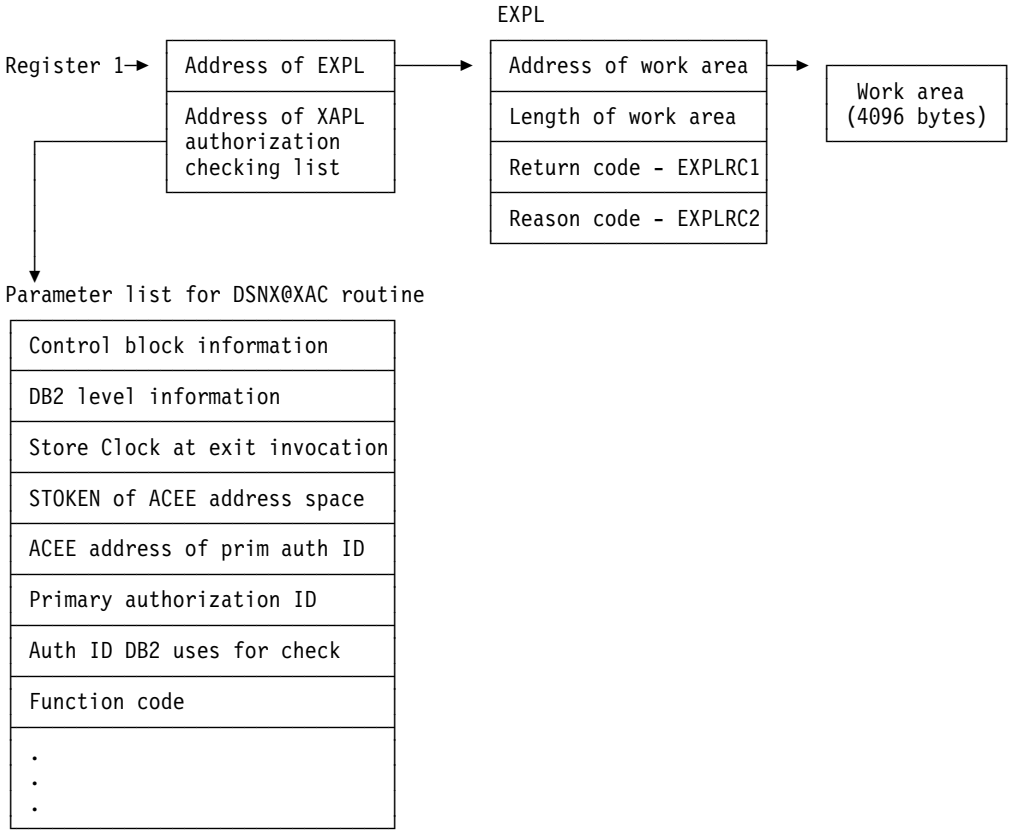


Figure 160. How an Authorization Routine's Parameter List Points to Other Information

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 X-76, and the authorization checking routine uses the standard exit parameter list (EXPL) described there. Table 112 shows the exit-specific parameter list, described by macro DSNDXAPL.

talign=left'.

Table 112 (Page 1 of 6). 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).

Table 112 (Page 2 of 6). 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
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: • Of the DB2 address space (*ssnm*DBM1) when XAPLFUNC is 1 or 3. • Of the primary authorization ID associated with this agent when XAPLFUNC is 2. 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: • When XAPLFUNC is 1 or 3, it contains the User ID of the DB2 address space (*ssnm*DBM1) • 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.

Table 112 (Page 3 of 6). 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
XAPLFUNC *	34	Signed, 2-byte integer	Input	Function to be performed by exit routine 1 Initialization 2 Authorization Check 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
XAPLTYPE	3E	Character,1	Input	DB2 object type: **D** Database **R** Table space **T** Table **P** Application plan **K** Package **S** Storage group **C** Collection **B** Buffer pool **U** System privilege
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. 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. The remaining 6 bits are reserved.

Table 112 (Page 4 of 6). 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
XAPLOBJN	40	Character, 20 bytes	Input	Unqualified name of the object with which the privilege is associated. It is one of the following names: **Name** / **Length** Database / 8 Table space / 8 Table / 18 Application Plan / 8 Package / 8 Storage Group / 8 Collection / 18 Buffer pool / 8 For special system privileges (SYSADM, SYSCTRL, and so on) this field might be blank. See macro DSNXAPRV. 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 113 on page X-42. 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 113 on page X-42. This parameter is left-justified and padded with blanks. If not applicable, it contains blanks or binary zeros.

Table 112 (Page 5 of 6). 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
XAPLREL2	7C	Character, 64 bytes	Input	Other related information. The contents of this parameter depends on the privilege being checked. See Table 113 on page X-42. 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	Input	Source of the request: **S** Remote request that uses DB2 private protocol. **' '** Not a remote request that uses DB2 private protcol. DB2 authorization restricts remote requests that use DB2 private protocol to the SELECT, UPDATE, INSERT and DELETE privileges.
XAPLRSV2	BF	Character, 15		Reserved
XAPLONWT	CE	Character, 1	Output	Information required by DB2 from the exit routine for the UPDATE and REFERENCES table privileges: **Value** — **Explanation** **' '** — Requester has privilege on the entire table ***** — Requester has privilege on just this column See macro DSNXAPRV for definition of these privileges.
XAPLDIAG	CF	Character, 40 bytes	Output	Information returned by the exit routine to help diagnose problems.

Table 112 (Page 6 of 6). 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
XAPLRSV3	F7	Character, 9		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 113.

Table 113 (Page 1 of 2). Related information for Certain Privileges

Privilege	Object Type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2
0053 (UPDATE) 0054 (REFERENCES)	T	Table Name Qualifier	Column Name if applicable	Database Name
0022 (CATMAINT CONVERT) 0050 (SELECT) 0051 (INSERT) 0052 (DELETE) 0056 (CREATE INDEX) 0061 (ALTER) 0073 (DROP) 0075 (LOAD) 0076 (CHANGE NAME QUALIFIER) 0097 (COMMENT ON) 0098 (LOCK) 0102 (CREATE SYNONYM) 0233 (ANY TABLE PRIVILEGE)	T	Table name qualifier	blank	Database name
0020 (DROP ALIAS) 0103 (ALTER INDEX) 0104 (DROP SYNONYM) 0105 (DROP INDEX)	T	Object name qualifier	blank	blank
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

Table 113 (Page 2 of 2). Related information for Certain Privileges

Privilege	Object Type (XAPLTYPE)	XAPLOWNQ	XAPLREL1	XAPLREL2
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

The data types and field lengths of the information shown in Table 113 on page X-42 is shown in Table 114.

Table 114. 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

Expected Output

Your authorization exit routine is expected to return certain fields when it is called. These output fields are indicated in Table 112 on page X-37. 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

Any other value returned in this field causes an abend. Register 3 points to EXPLRC1. If an abend occurs, the exit routine is not called again.

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

Any other value returned in this field causes an abend, and register 3 points to EXPLRC1. If an abend occurs, the exit routine is not called again. On authorization failures, the return code is included in IFCID 0140.

Handling Reason Codes

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 IFCID 0140.

Exit Abend

In the event that the exit routine abends, the authorization component of DB2 abends, and the exit is not called again. This applies to initialization, termination, and authorization checking.

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.

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.

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 Data in a Table Space or Partition” on page 2-63.

General Considerations

“General Considerations for Writing Exit Routines” on page X-74 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 X-76, and the edit routine uses the standard exit parameter list (EXPL) described there. Table 115 shows the exit-specific parameter list, described by macro DSNDEDIT. Figure 161 on page X-47 shows how the parameter list points to other row information.

Table 115. 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: 0 Edit-encode row for insert or update 4 Edit-decode row for retrieval
EDITROW	4	Address	Address of a row description. Its format is shown in Table 126 on page X-79.
	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 X-77.

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 or they are defined as NOT NULL WITH DEFAULT.

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.

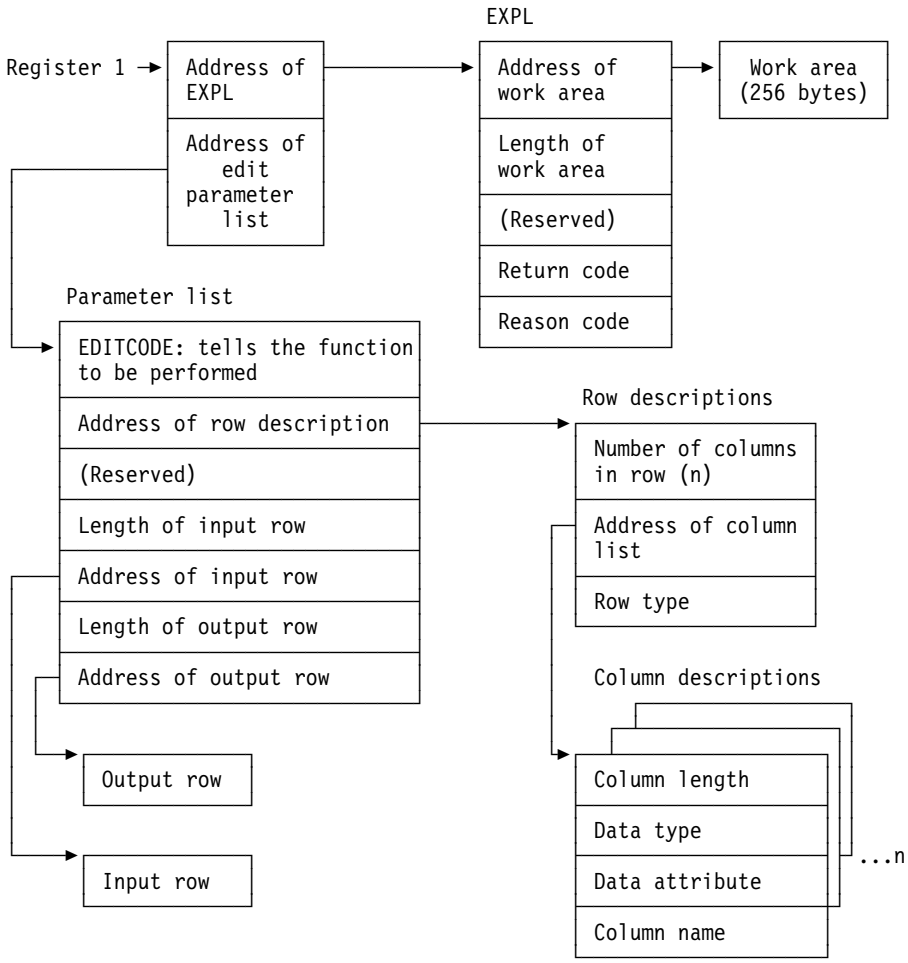


Figure 161. How the Edit Exit Parameter List Points to Row Information. The address of the nth column description is given by: RFMTAFLD + (n–1)×(FFMTE–FFMT); see "Parameter List for Row Format Descriptions" on page X-79.

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.

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 X-74 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 2-133. 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 X-76, and the validation routine uses the standard exit parameter list (EXPL) described there. Table 116 shows the exit-specific parameter list, described by macro DSNDRVAL.

Table 116. 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 126 on page X-79.
	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: 1 Insert, update, or load 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 X-77.

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 or they are defined as NOT NULL WITH DEFAULT.

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 162 shows how the parameter list points to other information.

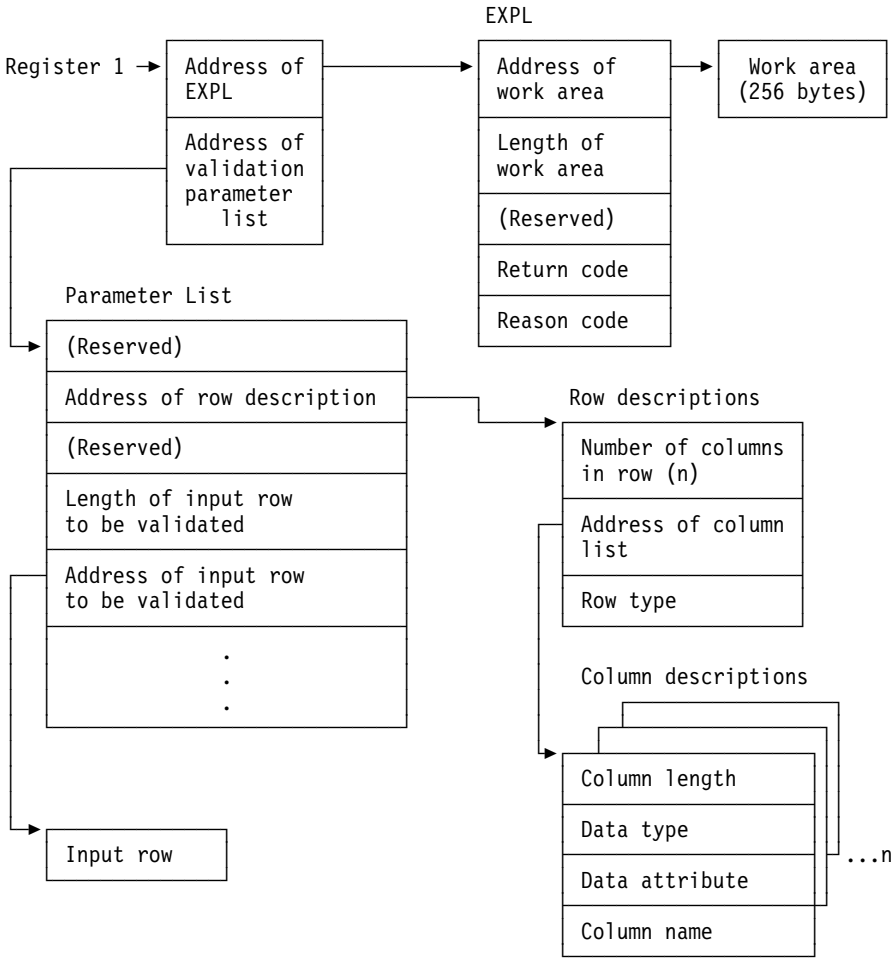


Figure 162. How a Validation Parameter List Points to Information. The address of the nth column description is given by: RFMTAFLD + (n–1)×(FFMTE–FFMT); see "Parameter List for Row Format Descriptions" on page X-79.

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 117. 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 Queries Sent to a Distributed System on page 2-49.

General Considerations

“General Considerations for Writing Exit Routines” on page X-74 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 Section 2 of *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 X-76, and the date or time routine uses the standard exit parameter list (EXPL) described there. Table 118 shows its exit-specific parameter list, described by macro DSNDDTXP.

Table 118. 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: 4 Convert from local format to ISO. 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
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 163 on page X-54 shows how the parameter list points to other information.

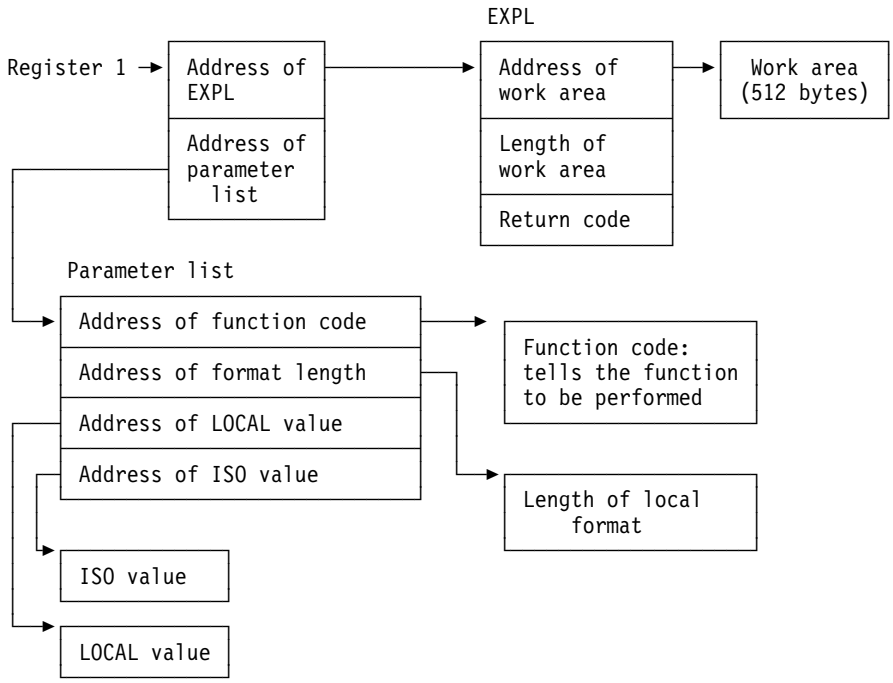


Figure 163. How a Date or Time Parameter List Points to Other Information

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 B of *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 X-74 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 X-76, 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 X-76. 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 119.

Table 119 (Page 1 of 2). 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: **Code** **Means** 20 VARCHAR 28 VARGRAPHIC
FPVDVLEN	2	Signed 2-byte integer	The maximum length of the string

Table 119 (Page 2 of 2). Format of String Value Descriptor for a Conversion Procedure

Name	Hex. Offset	Data Type	Description
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 X-77. 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 164 shows how the parameter list points to other information.

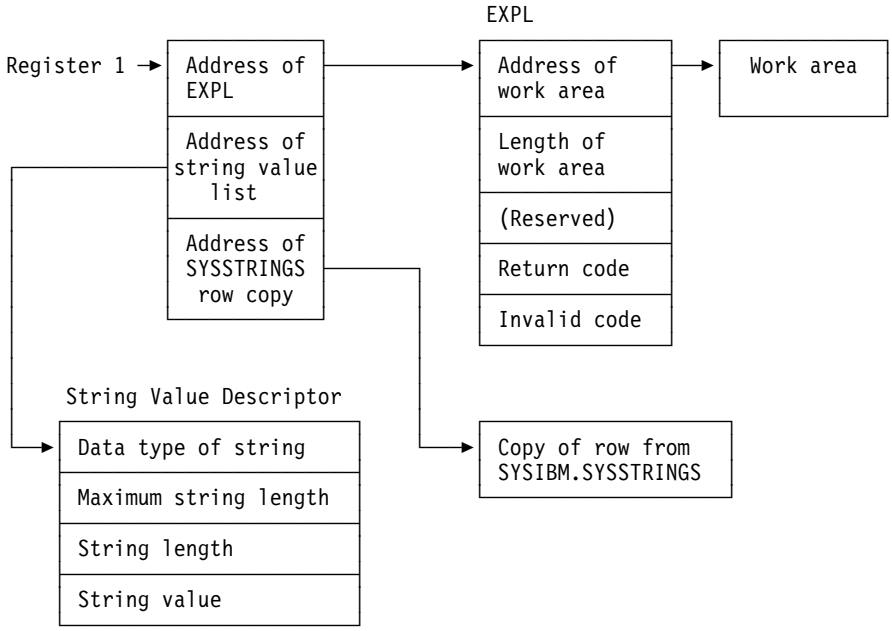


Figure 164. Pointers at Entry to a Conversion Procedure

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. But, 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 122 on page X-63. The length, precision, or scale of the encoded value must be compatible with its data type.

General Considerations

“General Considerations for Writing Exit Routines” on page X-74 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.

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:

Following that are the specific requirements for the three operations of field-definition:

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 165. Those areas are described in the following pages. The FPPL and the areas it points to are all described by the mapping macro DSNDFPPB.

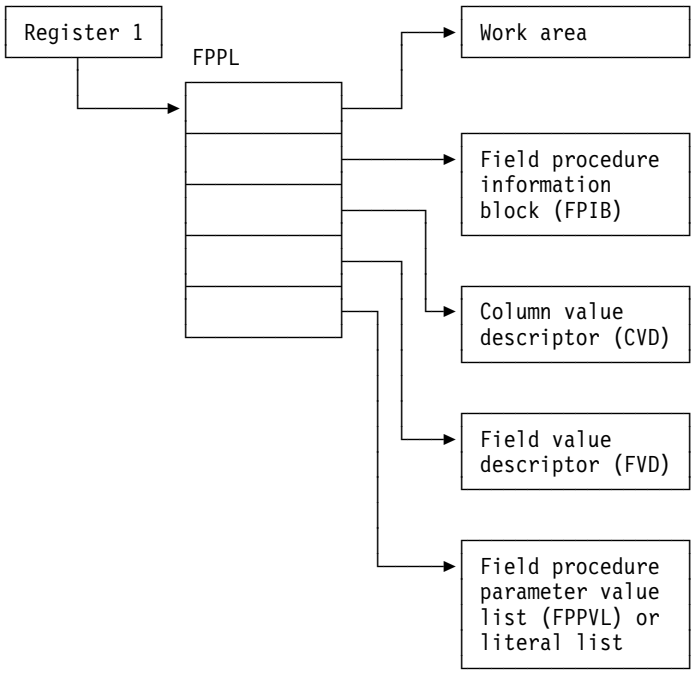


Figure 165. Field Procedure Parameter List

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 120.

Table 120. Format of FPIB, Defined in Copy Macro DSNDFPPB

Name	Hex. Offset	Data Type	Description
FPBFCODE	0	Signed 2-byte integer	Function code **Code Means** 0 Field-encoding 4 Field-decoding 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 121 on page X-62.

Table 121. 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: 1. A signed 4-byte integer giving the length of the following value descriptor, which includes the lengths of FPVDTYPE, FPVDLEN, and FPVDVALE. 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 122 on page X-63.

Table 122. Format of Value Descriptors

Name	Hex. Offset	Data Type	Description
FPVDTYPE	0	Signed 2-byte integer	Data type of the value: **Code** **Means** 0 INTEGER 4 SMALLINT 8 FLOAT 12 DECIMAL 16 CHAR 20 VARCHAR 24 GRAPHIC 28 VARGRAPHIC
FPVDVLEN	2	Signed 2-byte integer	• For a varying-length string value, its maximum length • For a decimal number value, its precision (byte 1) and scale (byte 2) • 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 165 on page X-60 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 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 122 on page X-63 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 165 on page X-60 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 122 on page X-63.
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 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 165 on page X-60 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 122 on page X-63.
FPVDVLEN	The length of the column value.
FPVDVALE	An area of unpredictable content that is as long as 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 Concern: 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 X-92.

General Considerations

"General Considerations for Writing Exit Routines" on page X-74 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 *MVS/ESA Programming: Authorized Assembler Services Guide* for additional material on SRB-mode processing.

  Performance Concern: 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 X-76, 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 123 shows the exit-specific parameter list; it is mapped by macro DSNDLOGX.

Table 123 (Page 1 of 2). 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
LOGXTYPE	10	Character, 1 byte	Situation identifier: I Initialization W Write T Termination P Partial control interval (CI) call
LOGXFLAG	11	Hex	Mode identifier. X'00' SRB mode 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: Address of first log buffer 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.

Table 123 (Page 2 of 2). Log Capture Routine Specific Parameter List

Name	Hex Offset	Data Type	Description
	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 Section 5 of *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 X-73 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 Section 2 of *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 Name
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 124 shows the contents of the list.

Table 124. 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	Contents
1	Address of pointer to the exit parameter list (shown in Table 125). *For a field procedure*, the address is that of the field procedure parameter list (see Figure 165 on page X-60).
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 166. The word following points to a second parameter list, which differs for each type of exit routine.

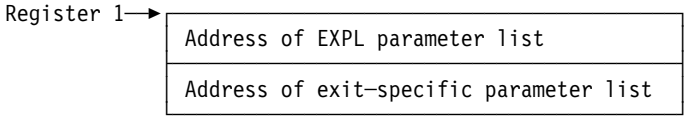


Figure 166. Use of Register 1 on Invoking an Exit Routine. (Field procedures and translate procedures do not use the standard exit-specific parameter list.)

The EXPL parameter list is shown below; its description is given by macro DSNDEXPL.

Table 125 (Page 1 of 2). 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: 2048 for connection and sign-on routines 512 for date and time routines and translate procedures (see Note 1). 256 for edit, validation, and log capture routines
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

Table 125 (Page 2 of 2). Contents of EXPL Parameter List

Name	Hex. Offset	Data Type	Description
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 all columns contiguously, regardless of word boundaries in physical storage.

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 127 on page X-79).

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	.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:

Col 1	Column 2 Column length (hex)		Column 3	Col 4
C01	0012	Information center	000030	A00

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.

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:

Col 1	Column 2 Column length (hex)			Column 3	Col 4
C01	0013	00	Information center	000030	A00

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 126 shows the general row description, and Table 127 shows the description of each column.

Table 126. 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 127.
RFMTTYPE	8	Character, 1 byte	Row type: X'00' = row with fixed-length columns X'04' = row with varying-length columns
	9	Character, 3 bytes	Reserved

Table 127. Description of a Column Format

Name	Hex. Offset	Data Type	Description
FFMTFLEN	0	Signed fullword integer	Column length attribute (see Table 128)
FFMTFTYP	4	Character, 1 byte	Data type code (see Table 128)
FFMTNULL	5	Character, 1 byte	Data attribute: X'00' = Null values are allowed. X'04' = Null values are not allowed.
FFMTFNAM	6	Character, 18 bytes	Column name

Table 128 (Page 1 of 2). 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

Table 128 (Page 2 of 2). Description of Data Type Codes and Length Attributes

Data Type	Code (FFMTFTYP)	Length Attribute (FFMTFLEN)
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

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)

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 xi.

This appendix discusses the following information about the log:

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 X-92

  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 X-96

  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 X-104

  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 types of log records, described under these headings:

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 Section 3 of *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.

Typical Unit of Recovery Log Records

Table 129 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 129. 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 130 shows the log records for processing and rolling back an insertion.

Table 130 (Page 1 of 2). 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

Table 130 (Page 2 of 2). Log Records Written for Rolling Back an Insertion

Type of Record	Information Recorded
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 131 summarizes the information logged for data and index changes.

Table 131. Information Logged for Database Changes

Operation	Information Logged
Insert data	The new row • On redo, the row is inserted with its original RID. • On undo, the row is deleted and the RID is made available for another row.
Delete data	The deleted row • On redo, the RID is made available for another row. • On undo, the row is inserted again with its former RID.
Update data	The old and new values of the changed data. • On redo, the new data is replaced • 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 offset in the index page, the new key value, and the data RID
Delete index entry	The offset in the index page, the deleted key value, and the RID of the data that was pointed to

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;[13] 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 Section 2 of *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 132 summarizes the information logged.

Table 132. 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 X-86 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.

[13] A page in a catalog table space that has links can contain up to 127 rows.

Database Page Set Control Records

These register several types of information, described below.

Page Set Allocate, Open, and Close: 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.

Exception States: Page set control records also register whether any database, table space, index space, or partition is in an exception state. An object can be in an exception state for any of these reasons:

- It is stopped.
- It is started for read-only (RO) or utility-only (UT) access. Read-write (RW) access is considered the normal condition.
- A write error range applies to it. If a write error occurs on a page, the page becomes unavailable. If more than one write error occurs, the error range spans all pages involved. All pages within the range are unavailable until the pages in error have been recovered.
- A DB2 utility has control of it. The object is unavailable to other processes.
- An image copy is required to make it recoverable.
- It is in check pending or recover pending 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 Section 3 of *Messages and Codes*.

Image Copies of Special Table Spaces: Image copies of DSNDB01.SYSUTILX, DSNDB01.DBD01, and DSNDB06.SYSCOPY are registered in the log, 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.

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 (4KB). 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 X-88.

Figure 167 on page X-87 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').

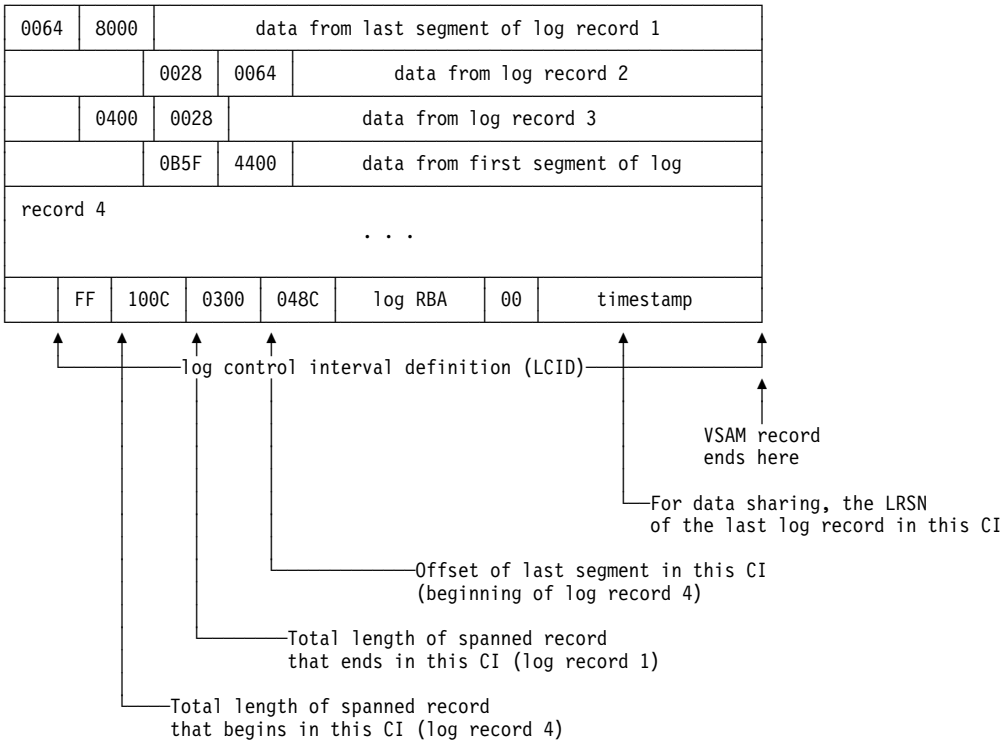


Figure 167. A VSAM CI and Its Contents

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 133 on page X-88.

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 133. Contents of the Log Record Header

Hex Offset	Length	Information
00	2	Length of this record or segment
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: B'00' A complete log record B'01' The first segment B'11' A middle segment 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[3]; 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 X-90 and “Log Record Subtype Codes” on page X-90.
[2] For a description of units of recovery, see “Unit of Recovery Log Records” on page X-82.

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 134.

Table 134 (Page 1 of 2). 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

Table 134 (Page 2 of 2). Contents of the Log Control Interval Definition

Hex Offset	Length	Information
07	6	Log RBA of the start of the CI
0D	1	Reserved
0E	7	Timestamp, reflecting the date and time the log buffer was written to the active log data set. The timestamp is the high order seven bytes of the Store Clock value (STCK).

Each recovery log record consists of two parts: a *header*, which describes the record, and *data*. Figure 168 shows the format schematically; the list below it describes each field.

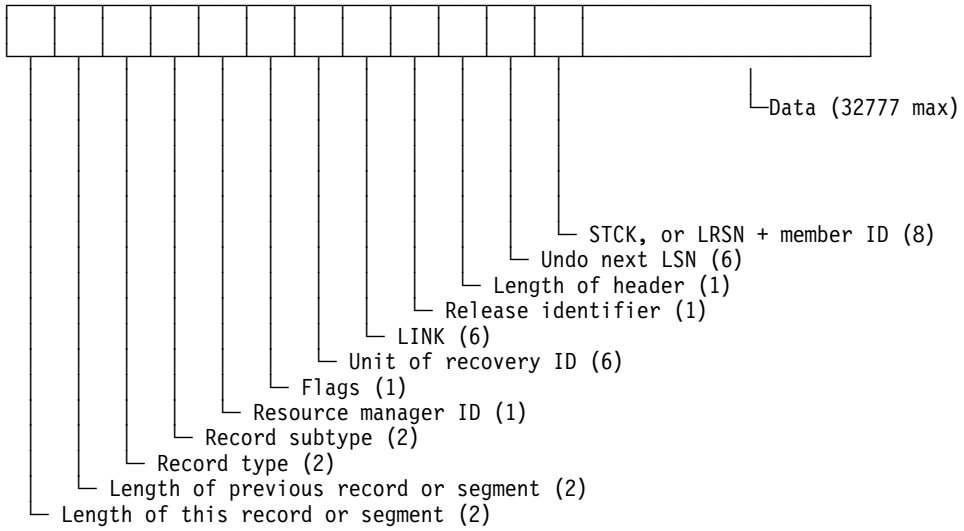


Figure 168. Format of a DB2 Recovery Log Record

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 X-90.
Subtype	Some types of recovery log records are further divided into subtypes. See “Log Record Subtype Codes” on page X-90.
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

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 Section 3 of *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 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

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. A list of DB2 mapping macros is provided in *Appendix H of Application Programming and SQL Guide* . 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

DB2 provides three methods of reading or capturing log records:

- Using the instrumentation facility interface (IFI). Your program can run only when DB2 is also running.
- Using the DSNJSLR macro and three stand-alone log services, OPEN, GET, and CLOSE. Your program can run even if DB2 is not also running.
- Using the log capture exit. Your program can run only when DB2 is also running.

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)" on page X-93
- "Reading Complete Log Data (IFCID 0306)" on page X-94

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 X-123.

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 8KB to 1MB in 4KB increments.

When the START TRACE command takes effect, from that point forward, until DB2 terminates, DB2 will begin writing 4KB 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 4KB log control intervals. This field is ignored when using the WQALLMOD=F option. For a complete description of the qualification area, see Table 147 on page X-131.

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 X-107.

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 X-93. IFCID 0306 must appear in the IFCID area. IFCID 0306 returns complete log records and the spanned record indicatiors 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.

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 X-123 for details.

Reading Log Records with OPEN, GET, and CLOSE

DB2 provides three offline 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 169 on page X-104 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, and passwords are provided. 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. If the data sets are password protected, the operator must supply a password when OPEN is executed.

Table 135 lists and describes the JCL DD statements used by stand-alone services.

Table 135 (Page 1 of 2). 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 X-100 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	
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. 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. (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.)

Table 135 (Page 2 of 2). JCL DD Statements for DB2 Stand-Alone Log Services

JCL DD Statement	Explanation
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. 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. 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. 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. 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. 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. 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 136.

Table 136. 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=YES|NO
                  ,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 136 on page X-99.
GPR 0	General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

See Section 4 of *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 136 on page X-99.
GPR 0	General-purpose register 0 contains a reason code associated with a nonzero return code in register 15.

See Section 4 of *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 4-7. Recovery Scenarios” on page 4-155.

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 137. 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 137 (Page 1 of 2). 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

Table 137 (Page 2 of 2). Stand-Alone Log GET Feedback Area Contents

Field Name	Hex Offset	Length (bytes)	Field Contents
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 136 on page X-99.
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 Section 4 of *Messages and Codes* for reason code details.

Sample Application Program Using Stand-Alone Log Services

Figure 169 on page X-104 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
   ⋮
*****************************************************************
*        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
   ⋮
   ⋮
*****************************************************************
*       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
   ⋮
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 169. Excerpts from a Sample Program Using Stand-Alone Log Services

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 X-92) 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 X-68 and refer to the previous sections of this appendix, “What the Log Contains” on page X-81 and “The Physical Structure of the Log” on page X-86.

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 xi.

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 170 shows the format of DB2 trace records.

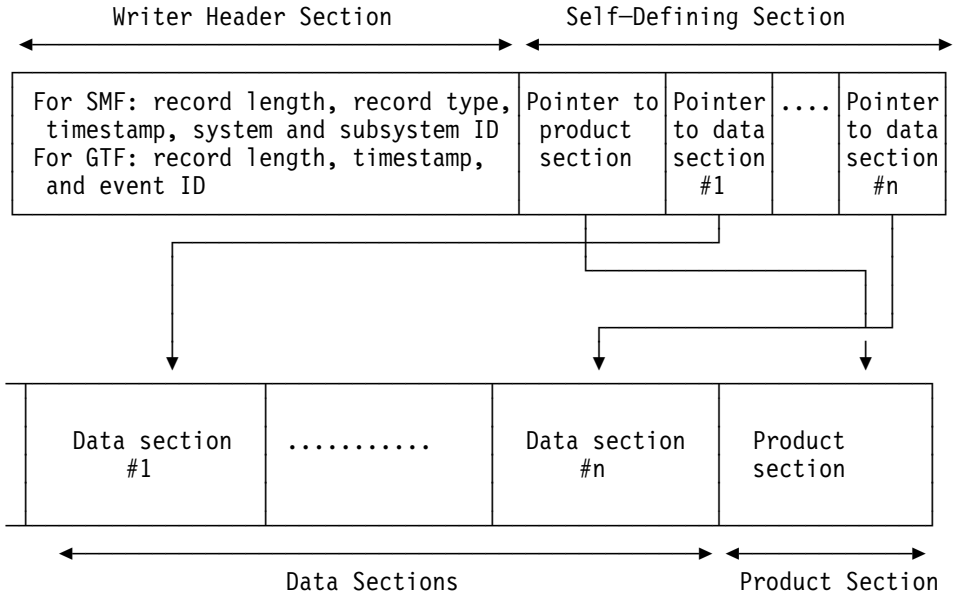


Figure 170. General Format of Trace Records Written by DB2

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 X-117. 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 *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	DSNDQW02
0201—0249 (except 0202, 230 and 239)	DSNDQW03
0202	DSNDQWS2, subtype=2
0230	DSNDQWST, subtype=3
0239	DSNDQWAS and DSNDQWA1
0250—0314	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 138 on page X-109. Figure 171 on page X-109 is a sample of the first record of DB2 performance trace output sent to SMF.

Table 138. Contents of SMF Writer Header Section

Hex Offset	Macro DSNDQWST, Statistics Field	Macro DSNDQWAS, Accounting Field	Macro DSNDQWSP, Monitor, Audit, & Performance Field	Description
0	SM100LEN	SM101LEN	SM102LEN	Total length of SMF record
2	SM100SGD	SM101SGD	SM102SGD	Segment descriptor
4	SM100FLG	SM101FLG	SM102FLG	System indicator
5	SM100RTY	SM101RTY	SM102RTY	SMF record type 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 171. DB2 trace output sent to SMF (printed with DFSERA10 print program of IMS)

Key to Figure 171	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

Key to Figure 171 on page X-109	Description
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
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 139.

Table 139 (Page 1 of 2). 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 0=Complete 2=Last 1=First 3=Middle

Table 139 (Page 2 of 2). Contents of GTF Writer Header Section

Offset	Macro DSNDQWGT Field	Description
31	QWGTDZZ2	Reserved
32	QWGTSSID	Subsystem ID
36	QWGTWSEQ	Sequence number
40	QWGTEND	End of GTF header

Figure 172 on page X-112 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 172. DB2 trace output sent to GTF (spanned records printed with DFSERA10 print program of IMS)

Key to Figure 172	Description
A 011C	Record length (field QWGTLEN); beginning of GTF writer header section

Key to Figure 172 on page X-112	Description
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 173 on page X-115 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 173 (Part 1 of 2). DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS)

```
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 173 (Part 2 of 2). DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS)

Key to Figure 173 on page X-115	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 174 on page X-118.

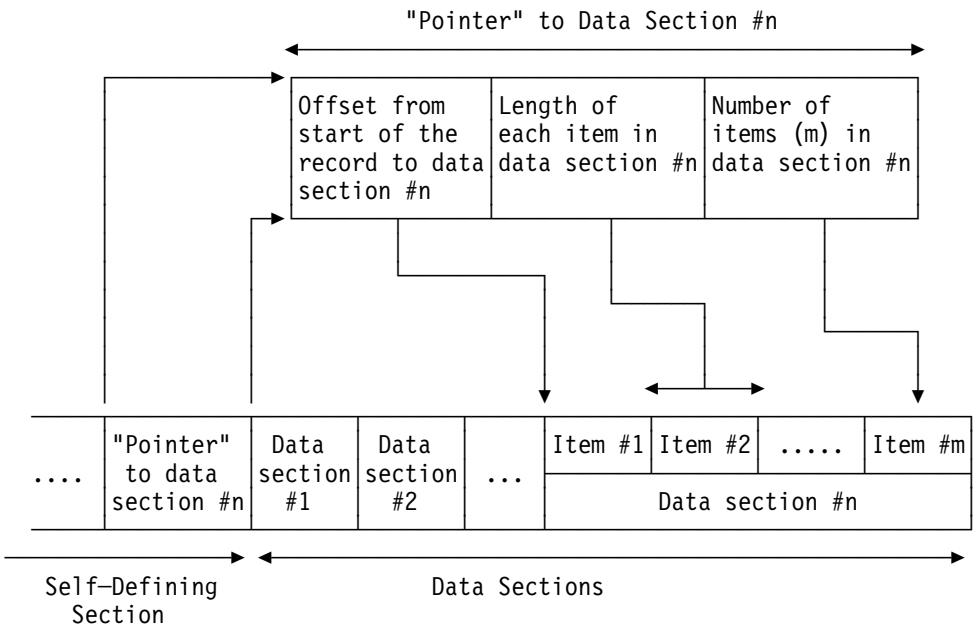


Figure 174. Relationship Between Self-Defining Section and Data Sections

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 140 (Page 1 of 2). 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

Table 140 (Page 2 of 2). Contents of Product Section Standard Header

Hex Offset	Macro DSNDQWHS Field	Description
44	QWHSLUUV	Uniqueness value
4A	QWHSLUCC	Commit count
4C	QWHSEND	End of product section standard header

Table 141. 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	QWHCATYPE	The type of system that is connecting
34	QWHCTOKN	Trace accounting token field
4A		Reserved
4C	QWHCEUID	End user's user ID at the user's workstation
5C	QWHCEUTX	End user's transaction name
7C	QWHCEUWN	End user's workstation name
8E	QWHCEND	End of product section correlation header

Table 142. 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 143 (Page 1 of 2). Contents of Distributed Data Header

Hex Offset	Macro DSNDQWHD Field	Description
0	QWHDLEN	Length of the distributed header
2	QWHDTYP	Header type
3		Reserved
4	QWHDRQNM	Requester location name
14	QWHDTSTP	Timestamp for DBAT trace record

Table 143 (Page 2 of 2). Contents of Distributed Data Header

Hex Offset	Macro DSNDQWHD Field	Description
1C	QWHDSVNM	EXCSAT SRVNAM parameter
2C	QWHDPRID	ACCRDB PRDID parameter
30	QWHDEND	End of distributed header

Table 144. 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 145. 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
4	QWHAMEMN	DB2 member name
C	QWHADSGN	DB2 data sharing group name
14	QWHAEND	End of header

Figure 175 on page X-121 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 00030931  00B3ADB8 E2E2D6D7
0005A0   A6E9BB31 E074C605  00000003 0000002D   00000002 E2C1D5E3  C16DE3C5 D9C5E2C1
                                                                      V
0005C0   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4   F1404040 A6E9BAEC  C4D90002 004C0200
0005E0   E2E8E2C1 C4D44040  E2E8E2C1 C4D44040   40404040 E3E2D640  40404040 C4E2D5C5
000600   E2D7D9D9 E2E8E2C1  C4D44040 00000007   00000000 00000000  00000000 00000000
                            W
000620   00000000 00000000  00341000 E2C1D56D   D1D6E2C5 40404040  40404040 A6E9BB2F
000640   38CCAC01 E2C1D56D  D1D6E2C5 40404040   40404040 C4E2D5F0  F3F0F1F0
```

Figure 175. DB2 Distributed Data Trace Output Sent to SMF (printed with DFSERA10 print program of IMS). 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 175	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 listed in Appendix H of *Application Programming and SQL Guide*.

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 xi.

This appendix helps you to use the instrumentation facility interface (IFI). The following topics are discussed in this appendix:

| IFI can be accessed through any of the DB2 attachment facilities, including the
| Resource Recovery Services (RRS) of MVS.

IFI uses the standard security mechanisms that DB2 uses—connection authorization, plan authorization, and so forth. For more information about security, see “ Section 3. Security and Auditing” on page 3-1. 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 X-177, 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.

What IFI Can Do

The DB2 instrumentation facility gathers trace data that can be written to one or more destinations that you specify. IFI is designed for a program needing *online* information about DB2. 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” on page X-124
- “Obtaining Trace Data” on page X-124
- “Passing Data to DB2 through IFI” on page X-125

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
  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 X-81.
- To retrieve SQL changes synchronously from the log for processing in an application. See “Reading Log Records” on page X-92 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 X-126.

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 X-129.

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 X-140.

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 X-142.

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”), 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 X-143.

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 TSO DSNELI
CICS DSNCLI IMS DFSLI000
RRSAF DSNRLI

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 Section 6 of *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. But, 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 X-180.

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 :hdref refid=dxexits. for 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, 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 X-144 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 X-146 for a description of the return area.

output-area

Contains the varying-length command. See “Output Area” on page X-147 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 X-141.

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 146. 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.
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 176. 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, 0202, 0230, 0254 0306, 0316, and 0317. For a description of the data these records provide, see “Synchronous Data” on page X-137. 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 X-144 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 X-93.
- 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 X-81.
- 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 X-146 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 X-146 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 X-137 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 X-135 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 147.

Table 147 (Page 1 of 5). 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: • WQALLN21. When specified, the location name qualifications (WQALLOCN and WQALLUWI) are ignored. • WQALLN22. When specified, the location name qualifications (WQALLOCN and WQALLUWI) are used. • WQALLN23. When specified, the log data access fields (WQALLTYP, WQALLMOD, WQALLRBA, and WQALLNUM) are used for READS calls using IFCID 129. • WQALLN4. When specified, the location name qualifications (WQALLOCN and WQALLUWI), the group buffer pool qualifier (WQALGBPN) and the read log fields are used. # • WQALLN5. When specified, the dynamic statement # cache fields (WQALFFLD, WQALFVAL, WQALSTNM, # and WQALSTID) are used for READS calls for IFCID # 0316 and 0317.
	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.
| WQALCORR	30	Character, 12 bytes	Correlation ID; binary zero if it is not to be used.

Table 147 (Page 2 of 5). Qualification Area Fields. This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex Offset	Data Type	Description
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.
	62	Hex, 2 bytes	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 147 (Page 3 of 5). Qualification Area Fields. 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: • 'D' means to return the direct log record specified in WQALLRBA if the IFCID is 0306. • 'F' means to 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. • 'R' means to access the CI as specified by the value in the WQALLRBA field. If the requested number of CIs (as specified in WQALLNUM) is not currently available, no data is returned and a reason code is returned to the IFCA. • 'H' means to return the highest LRSN or log RBA in the active log. The value is returned in the field IFCAHLRS in the IFCA. • 'N' means to return the next set of qualified log records. • 'T' means to terminate the log position that is held to anticipate a future mode 'N' call. • 'P' means that 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): • '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. • 'Y' indicates that a data description will be returned for each table in the list for every new request. • '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. • If the IFCID is 0306, this is the log RBA or LRSN to be used in mode 'F'.
WQALGBPN	9C	Character, 8 bytes	Group Buffer Pool Qualifier
WQALLCRI	A4	Hex, 1 byte	Log Record Selection Criteria • '00' indicates the return DB2CDC and UR control log records.

Table 147 (Page 4 of 5). Qualification Area Fields. This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex Offset	Data Type	Description
WQALLOPT	A5	Hex, 1 byte	Processing Options relating to decompression • '01' indicates to decompress the log records if they are compressed. • '00' indicates that decompression should not occur.
WQALFLTR	A6	Hex, 1 byte	For an IFCID 0316 request, identifies the filter method: • X'00' indicates no filtering. This value tells DB2 to return information for as many cached statements as fit in the return area. • 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. • 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. For an IFCID 0306 request, indicates whether DB2 merges log records in a data sharing environment: • X'00' indicates that DB2 merges log records from data sharing members. • X'03' indicates that DB2 does not merge log records from data sharing members.

Table 147 (Page 5 of 5). Qualification Area Fields. 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: • 'E' - the number of executions of the statement (QW0316NE) • 'B' - the number of buffer reads (QW0316NB) • 'G' - the number of GETPAGE requests (QW0316NB) • 'R' - the number of rows examined (QW0316NR) • 'P' - the number of rows processed (QW0316NP) • 'S' - the number of sorts performed (QW0316NS) • 'I' - the number of index scans (QW0316NI) • 'T' - the number of table space scans (QW0316NT) • 'L' - the number of parallel groups (QW0316NL) • 'W' - the number of buffer writes (QW0316NW) • 'A' - the accumulated elapsed time (QW0316AE). 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. 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, 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.
WQALSTID	BC	Unsigned 4-byte integer	For an 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. To obtain this ID, issue a READS request for IFCID 0316. The ID is in field QW0316TK. This field and WQALSTNM uniquely identify a cached statement.

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 148. Qualification Fields for IFCIDs

These IFCIDs...	Are allowed to use these qualification fields
0124, 0147, 0148, 0150	WQALACE WQALAIT2 WQALPLAN[1] WQALAUTH[1] WQALOPID[1] WQALCONN[1] WQALCORR[1] WQALASID WQALLUWI[1] WQALLOCN[1] WQALD147
0149	WQALREST WQALHASH
0129	WQALLTYP WQALLMOD WQALLRBA WQALLNUM
0185	WQALCDCD
0254	WQALGBPN
0306	WQALFLTR WQALLMOD WQALLRBA WQALLCRI WQALLOPT
0316	WQALFLTR WQALFFLD WQALFVAL
0317	WQALSTNM 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'.

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 "Parallel Operations and Query Performance" on page 5-299 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 *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 X-81.

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).

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” on page X-139.

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 X-144 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, refer to data set *prefix*.SDSNSAMP(DSNWMSGS), which is shipped to you as part of the product. See also “DB2 Trace” on page X-177 and “Appendix D. Interpreting DB2 Trace Output” on page X-107 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” 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 147 on page X-131.
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 147 on page X-131.

   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.

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 X-143 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 X-146 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, we strongly recommended that you 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 X-149.

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 X-177.

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 146 on page X-128, 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 X-144 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 X-147 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 149) 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 149 for IFCIDs that can be used by the write function.

Table 149. 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 X-146 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” on page X-144, below
“Return Area” on page X-146
“IFCID area” on page X-146
“Output Area” on page X-147

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 150.

Table 150 (Page 1 of 2). 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.
	2	Hex, 2 bytes	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 Section 4 of *Messages and Codes* for information about reason codes.
IFCARC2	10	Four-byte signed integer	Reason code for the IFI call. Binary zero indicates a successful call. See Section 4 of *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.
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.
IFCAOPWS	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.

Table 150 (Page 2 of 2). Instrumentation Facility Communication Area. The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex Offset	Data Type	Description
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.
IFCADD	64	Character, 80 bytes	Diagnostic information. • IFCAFCI, offset 64, 6 bytes 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 X-93 for more information. • IFCAGBPN, offset 74, 8 bytes This is the group buffer pool name in error if IFCARC2 is 00E60838 or 00E60860 • IFCABSRQ, offset 88, 4 bytes This is the size of the return area required when the reason code is 00E60864. • IFCAHLRS, offset 8C, 6 bytes 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').

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 151. 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.

Table 152. 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 X-81

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 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 153. 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'. 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)  '
```

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. Refer to *prefix*.SDSNSAMP(DSNWMSGS) for the format of the trace record and its header.

- 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*.SDSNSAMP(DSNWMSGS).

For detailed information about the format of trace records and their mapping macros, see “Appendix D. Interpreting DB2 Trace Output” on page X-107, or Appendix H of *Application Programming and SQL Guide*.

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 177. 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 X-177 and “Appendix D. Interpreting DB2 Trace Output” on page X-107.

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 178. 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 5-7. Improving Concurrency” on page 5-137.

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 Section 4 of *Messages and Codes*.

Appendix F. Sharing Read-Only Data

DB2 allows you to read common DB2 data from a number of DB2 subsystems using *shared read-only data*. With shared read-only data, you can share the physical data for a given database among DB2 subsystems, but one DB2 subsystem has exclusive control over updating the data in that shared database. The database in the system that can update is called the *owner*. The same database in any other system is called a *reader*. A system can only read the data in a reader. Data cannot be read from a reader while the owner is being updated.

Occasionally, we use the term “owning DB2” to refer to the DB2 subsystem containing the owner. We also use the term “read-only DB2” or “reading DB2” to refer to the DB2 subsystem containing a reader.

We recommend that you use full read-write data sharing, rather than shared read-only data. For information on full read-write data sharing, see *Data Sharing: Planning and Administration*.

Overview of Shared Read-Only Data

- “Implementing Shared Read-Only Data” on page X-155

  This section gives an overview of the entire process of setting up global resource serialization (GRS), creating objects in the owner and the readers, and running utilities.

- “Create DB2 Objects to be Shared” on page X-158

  You do not create any physical data objects for the reader, but you do perform data definitions for it. These make the DB2 catalog for the reader mimic the catalog for the owner. This allows applications running on the read-only DB2 to access the physical data belonging to the owner.

- “Starting and Stopping a Shared Database” on page X-164

  This is an important topic to understand before you begin working with shared read-only data. To update data in an owner, you must first stop any relevant index and table spaces in the readers.

- “Maintaining Shared Read-Only Data” on page X-167

  This section describes how you can update data, and add, alter or drop objects in a shared database. It also describes recovery considerations and special considerations for running utilities in a shared read-only data environment.

Prerequisites for Shared Read-Only Data

The sharing DB2 subsystems can be on the same processor, or on separate processors.

If the subsystems are on separate MVS systems or processors, you must tell the Data Facility Product that the integrated catalog facility catalog is shared. See *DFSMS/MVS: Access Method Services for the Integrated Catalog* for information about how to do that.

The shared data must be stored on DASD that is channel-connected to the subsystems that access the data. Data integrity is enforced through MVS's global resource serialization (GRS), or an equivalent, and VSAM sharing options.

If you are using GRS, the processors must communicate with dedicated channel-to-channel connections; in other words, you cannot use the same channel connections as you would use for the distributed data facility.

Benefits of Shared Read-Only Data

If you have a substantial amount of common shared data, and you can tolerate the operational complexity, consider shared read-only data in the following situations:

- You need faster performance than you can get by distributing data, and you do not need all the function the distributed data facility offers. See Section 3 of *Installation Guide* for more information about the distributed data facility. See “Comparing Shared Read-Only Data and Distributed Data” on page X-155 for a comparison of the distributed data facility and shared read-only data.
- You want to conserve DASD. Without shared read-only data, if you want to share your data across DB2 subsystems, you must replicate the data in all the sharing systems. With shared read-only data, you only need one copy of the data.
- You need to improve capacity throughout the system.

  By sharing the query processing load among more than one DB2 subsystem, you ease the load on the query applications during prime work hours. By running update transactions in a batch window—without the additional overhead of the query processing—you improve the capacity of the update transactions.

  Alternatively, you can run daily transactions that update the data in one owning subsystem, and then run batch data analysis and reports at night on several readers.

Costs of Shared Read-Only Data

We recommend that you carefully consider the following drawbacks to shared read-only data:

- Shared read-only data is operationally complex. You must stop index and table spaces or partitions before you can update the data, and you must restart them again to read data on multiple DB2 subsystems.

  Consider using NetView CLISTs to ensure that commands are entered in the correct sequence on the correct subsystems. Also, you must devise a method to make sure that definitions for the owner are also created for the readers.
- The data is not available for update continuously. Shared read-only data does not allow for concurrent read/write access.
- In order for a nonpartitioned index to be connected in read-only (RO) mode, all logical partitions must be started for read-only access.

Comparing Shared Read-Only Data and Distributed Data

Both the distributed data facility and shared read-only data give you ways to access data outside a single DB2 environment. The following table summarizes the main differences between the two methods:

Table 154. Comparing Shared Read-only Data and DDF

	Shared read-only data	**DDF**
Update flexibility	Not flexible. There is a strict window for update activity.	More flexibility for update.
Physical proximity	Must be close enough for channel-to-channel connections between DB2s.	With proper connections, distance is not a problem (except, perhaps, for performance).
DASD	All participating DB2s need access to shared DASD.	DASD does not have to be shared.
Setup and operational complexity	GRS setup can be complicated if GRS expertise is lacking. Procedures for starting and stopping objects are complex.	VTAM configurations must be established, and VTAM expertise is needed.
Performance	No real difference from normal DB2 performance.	Network overhead can be high, especially for single selects and updates. Block fetch can alleviate overhead for large selects with no intention for updating.
Accounting	All DB2 activity in the readers is collected in the SMF log of the MVS system on which the DB2 subsystem resides.	DB2 activity records are cut in both the requesting and the serving subsystems—they must be combined for proper charging.

Using DDF and Shared Read-Only Data Together: You can use a DDF and a shared read-only data configuration for the same shared data. For example, you might want a non-DB2 requester to have occasional access to the shared data. However, be aware that operating such a configuration can become very complex.

Implementing Shared Read-Only Data

To implement your setup for shared read-only data, first make sure your GRS environment is ready for use with DB2. Second, plan how you will set up and maintain data definitions for the owner and the readers. Then, create the owner, either by altering an existing database to be shared or by defining a new one and loading it with data. Finally, create the counterpart objects (with no data) in the reader.

Steps for Sharing an Existing Database

1. “Tune GRS for DB2” on page X-157.
2. “Plan to Set Up and Maintain Data Definitions” on page X-156.
3. “Alter an Existing Database to be Shared” on page X-157.
4. Create Objects for the Readers.

   See “Create DB2 Objects to be Shared” on page X-158.
5. Re-create statistics in the readers.

You do this by performing RUNSTATS on all DB2s for shared objects, as described in “Appendix G. Using Tools to Monitor Performance” on page X-173. Or, you can use SQL to update the statistics in the DB2 catalog for each reader. The columns you can update are shown in Table 86 on page 5-244.

Steps for Sharing a New Database

1. “Tune GRS for DB2” on page X-157.
2. “Plan to Set Up and Maintain Data Definitions.”
3. Create Objects for the Owner.

   See “Create DB2 Objects to be Shared” on page X-158.
4. “Load Data in the Owner” on page X-163.
5. Create Objects for the Readers.

   See “Create DB2 Objects to be Shared” on page X-158.
6. Re-create statistics in the readers.

   You do this by performing RUNSTATS on all DB2s for shared objects, as described in “Appendix G. Using Tools to Monitor Performance” on page X-173. Or, you can use SQL to update the statistics in the DB2 catalog for each reader. The columns you can update are shown in Table 86 on page 5-244.

Plan to Set Up and Maintain Data Definitions

If you are creating a new database to be shared, make sure that you do not have any existing databases, table spaces, or index spaces on a sharing DB2 that have the same fully qualified name as those you want to create for sharing.

Although there is only one physical copy of the DB2 objects in a shared database, the DB2 catalogs for the readers must also contain information about the objects. Thus, you must create counterparts in all readers for all, or a subset of, the owner database: table spaces, indexes, and tables.

Optionally, you can create counterparts for authorizations, views, synonyms, and aliases. The decision to re-create this information depends on whether your application plans and packages need this information to be the same. You must re-create the catalog entries for any item that an application running against a reader makes use of.

Recommendation: Because you must keep DB2 catalog information the same for both readers and owners, we recommend that you keep all data definition statements in a centralized location. Saving the data definition statements makes the task of re-creating those definitions much easier.

Anything you drop or alter in the owner must also be dropped or altered in the readers. The reader must not contain any objects that are not also in the owner.

Tune GRS for DB2

We recommend that someone experienced with GRS perform this step. To create DB2 sharing in an existing GRS complex (cross-MVS sharing), specify VSAM SHAREOPTIONS (1,3) in the access method services DEFINE statement for the VSAM data sets for table spaces and index spaces and the integrated catalog facility catalog. For cross-MVS sharing, the “3” means the data set can be used by multiple users, with each user responsible for maintaining data integrity. For shared read-only data, DB2 relies on GRS to maintain data integrity.

See “Managing Your Data Sets Using Access Method Services” on page 2-69 for information about creating data sets with access method services. If you use DB2 storage groups, this is taken care of automatically.

Excluding Data Sets

For read-only data sharing, you must exclude from sharing the bootstrap data set, the catalog and directory, and all user data sets that are not shared resources. Also exclude the MVS system catalog. To exclude these VSAM data sets from sharing, add entries to your GRS SYSTEMS exclusion resource name list. Use TYPE(GENERIC) entries with QNAME(SYSDSN) and RNAME(data set high level qualifier).

If you enable full data sharing at a later time, you must remove the bootstrap data set, the catalog, and the directory from the exclusion resource name list.

If you have too many VSAM data sets to exclude (perhaps because your naming convention doesn't make it convenient), you can create a unique high level qualifier for shared read-only data sets. You can modify the GRS ISGGREX0 exit at entry point ISGGSEEX to allow only the data sets with the new unique high level qualifier to participate in GRS sharing. For information about changing the high level qualifier, see “Changing the High-Level Qualifier for DB2 Data Sets” on page 2-139. For information about the ISGGREX0 exit, see *MVS/ESA Installation Exits*. See *MVS/ESA Planning: Global Resource Serialization* for more information about planning for and implementing global resource serialization.

Alter an Existing Database to be Shared

If you have SYSADM or SYSCTRL authority, you can convert the status of a database from non-shared to owner. However, when you have created a reader database, you cannot use SQL to convert its shared status. Instead, you must drop the database and redefine it. For information about altering or dropping a database, see “Altering DB2 Databases” on page 2-125.

Converting a Non-Shared Database to Sharing Owner:

To convert a database from non-shared to owner, do the following:

1. Make sure all user-managed data sets for table spaces and indexes are defined with VSAM SHAREOPTIONS(1,3). If they are not, use access method services to alter the SHAREOPTIONS values for all existing user-managed data sets in the database. (DB2-managed data sets are converted when you alter the database.)

2. Use the DISPLAY DATABASE command to make sure there is no check pending or recover pending status set for the objects in the database.
3. Make sure no users are accessing the database by entering the following command:

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

4. Start the database for read-write access by entering the following command:

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

5. Enter the ALTER DATABASE statement and specify ROSHARE OWNER:

   ```
   ALTER DATABASE NEWDB
     ROSHARE OWNER;
   ```

6. Stop the database by entering the following command:

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

   This make the VSAM SHAREOPTIONS effective on the owner's storage group's data sets.

Create DB2 Objects to be Shared

The information under this heading, up to “Starting and Stopping a Shared Database” on page X-164, is General-use Programming Interface and Associated Guidance Information, as defined in “Notices” on page xi.

Create DB2 Storage Groups

You can use user-defined or DB2-defined data sets with shared read-only data. This section describes specific information about creating DB2 storage groups for DB2-defined data sets in a shared read-only environment. General information about creating storage groups is described in “Implementing Your Storage Groups” on page 2-82. The VCAT and password values should be the same for both the reader and owner. The volumes clause does not have to be the same, but having it the same allows users on a reading DB2 subsystem to query the catalog for volume information.

When creating objects on the owning DB2 subsystem, make sure the selected volumes are on shared DASD.

When you create the DB2 storage group, specify the serial numbers of the volumes needed for the table space or index. When a data set is defined in the shared read-only environment, all volume serial numbers from the DB2 storage group are stored in the VSAM catalog and used for data set extension.

If you are using SMS to manage your data sets, use an asterisk to specify each volume you expect to use. For example, if you anticipate needing three volumes to extend the data set, list three asterisks in the VOLUMES clause of your CREATE STOGROUP statement:

```
CREATE STOGROUP stogroupname (VOLUMES('*','*','*')...
```

For each asterisk, SMS uses a volume from the storage group you defined for SMS.

If you do not list enough volumes, data set extension fails. In that case, you can alter the storage group to contain more volumes, using the ALTER STOGROUP statement. New volumes added to a DB2 storage group are not available to a data set until you do one of the following:

- Execute a CREATE TABLESPACE or CREATE INDEX statement
- Run the LOAD REPLACE, REORG, or RECOVER utility

Unlike the case with normal DB2-managed data sets, when you create a table space in the reader that has the same name as one in the owner, DB2 does not delete and redefine the data set.

Create a Database

Use the ROSHARE clause in the SQL CREATE DATABASE statement to define a database as shared and to specify either that it is an owner (which can update data), or is a reader (which can only read, not update, data). To use the ROSHARE clause, you must have SYSADM or SYSCTRL authority.

After you create a database, the ROSHARE column in the SYSIBM.SYSDATABASE catalog table indicates whether a database is an owner or reader (or neither) and includes the time when the owner was created. The catalog entry for the reader is not updated with this timestamp until you create the first table space in the reader.

For ease of use, we recommend that you use the same storage group attributes for both the owner and the readers and that you make the buffer pool page size attributes compatible. For example, specify both buffer pools as 4KB pages, or both buffer pools as 32KB pages.

Create the Owner:

Issue CREATE DATABASE with the ROSHARE OWNER option in the owning subsystem. The following example creates a database named DONSDB and makes it an owner:

```
CREATE DATABASE DONSDB
          ROSHARE OWNER;
```

Create the Readers:

Issue CREATE DATABASE with the ROSHARE READ option in the each reading subsystem. The following example creates DONSDB and defines it as read-only.

```
CREATE DATABASE DONSDB
          ROSHARE READ;
```

You must not create any objects for the readers that differ from the objects in the owner.

Create Table Spaces

You can create all types of table spaces (simple, segmented, and partitioned) in a shared database. After creating a table space in the owner, you can use the same data definition statement to create the counterpart table space in the readers, or you can specify different values for some options.

For both DB2-managed data sets and user-managed data sets, be sure the data sets are on shared volumes (use the same integrated catalog facility catalog name) that are accessible to both owning and reading DB2 subsystems.

Create Table Spaces for the Owner

1. If you are managing your own data sets, define all data sets for the owner's shared table spaces with VSAM SHAREOPTIONS(1,3), which allows either multiple readers *or* a single updater at any given time. If you let DB2 create data sets, this is taken care of automatically.
2. Issue CREATE TABLESPACE in the owning DB2 subsystem. The following example creates a simple table space in the shared database called DONSDB:

```
CREATE TABLESPACE TS1
       IN DONSDB
       BUFFERPOOL BP0
       LOCKSIZE TABLESPACE
       CLOSE NO
       USING VCAT DONSCAT;
```

Create Table Spaces for the Readers

1. Stop the table space in the owner by entering the following command for the owner:

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

   This writes the changes to DASD and closes the data sets. More information about the STOP DATABASE and START DATABASE commands for shared databases is included in “Starting and Stopping a Shared Database” on page X-164.

   Optionally, if you want read access to the owner's data, enter the following command:

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

2. Issue your CREATE TABLESPACE statement in the reading DB2 subsystem, specifying the same values as in the owner for these options:

   NAME IN
   USING NUMPARTS
   SEGSIZE DSETPASS

   You can specify different values for the following options:

   - BUFFERPOOL

     Although you can specify a different buffer pool for the reader, both buffer pool values must have the same page size attribute (4KB pages, for example).

   - CLOSE

     With shared read-only data, you usually control the closing of data sets using the STOP command. However, you can use the CLOSE option to determine the order in which data sets are closed in those cases when DB2 has to close data sets to control the number of open data sets. See “CLOSE Clause” on page 2-89 for more information about the CLOSE option.

   - LOCKSIZE

For all table spaces accessed in read-only mode (including non-shared table spaces), DB2 implicitly uses LOCKSIZE TABLESPACE when accessing table spaces in read-only mode. DB2 honors the LOCKSIZE attribute in the owner when a table space is accessed in read-write mode. For more information about locking, see “Chapter 5-7. Improving Concurrency” on page 5-137.

- FREEPAGE and PCTFREE

  It does not matter what values you specify for FREEPAGE and PCTFREE in the reader because you cannot load and reorganize data in that database.

- COMPRESS

  It does not matter what you specify for COMPRESS in the reader because compression only occurs with UPDATE and INSERT SQL statements, or with the LOAD and REORG utilities, none of which you can perform on the reader. Because the data sets themselves contain information about compression, DB2 processes compressed data correctly for the read-only queries.

Create Tables

To create shared tables, first define the tables in the owner and then create their counterparts in the readers. To create a counterpart table, you must append the object identifier (OBID) of the table in the owner to your data definition statement.

Create Tables for the Owner

1. Stop the table space for the table in all readers and start it read-write in the owner.
2. Issue the SQL CREATE TABLE statement in the owning DB2 subsystem. The following example creates a table named DON.EMP in DONSDB.TS1.

```
CREATE TABLE DON.EMP
       (EMPNAME CHAR(18),
        EMPID CHAR(8) NOT NULL,
        EMPATTR CHAR(8))
       IN DONSDB.TS1;
```

Create Tables for the Readers

1. Stop the table space for the table in the owner.
2. Find the OBID for the table in the owner by querying SYSIBM.SYSTABLES in the owning DB2 subsystem. The following example queries the OBID of DON.EMP.

```
SELECT OBID
       FROM SYSIBM.SYSTABLES
       WHERE CREATOR='DON' AND
       NAME='EMP';
```

3. After you know the OBID, use that identifier in the CREATE TABLE statement in the reading subsystem, as shown below.

```
CREATE TABLE DON.EMP
       (EMPNAME CHAR(18),
        EMPID CHAR(8) NOT NULL,
        EMPATTR CHAR(8))
       IN DONSDB.TS1
       OBID 14;
```

You must specify the same values in the reader and the owner for the following options:

- The IN clause, which *must* be specified
- The column data types
- The order of the columns
- The edit procedure, if you use one.

You will probably want to keep the table name and column names the same, as well, so that the systems can share plans and packages.

Consider using the same FIELDPROCs in the reader and owner tables to maintain a consistent look in all sharing subsystems. The VALIDPROC, AUDIT CHANGES and DATA CAPTURE clauses are of no use in a reader, because the reader cannot be updated.

Using Referential Constraints

There are no restrictions for enforcing referential integrity in a shared database, unless the referential structure crosses database boundaries. Two tables participating in a referential relationship across databases must either be:

- Updateable, as when both tables are in owners or one table is in an owner, and the other is in a non-sharing database
- Read-only, which means both tables are in readers.

As with regular non-sharing databases, when a parent table resides in one database and a dependent table resides in a different database, then you must define both databases before creating the referential constraint.

For more information about referential integrity, see “Chapter 2-3. Maintaining Data Integrity” on page 2-19.

Re-creating Referential Relationships in the Reader: Optionally, you can re-create referential relationships in the DB2 catalog in the reading DB2 subsystem so that users can query that catalog and view the referential relationships. RELOBID1 and RELOBID2 contain the value zero, because no object descriptors are generated for the reader. The reading DB2 subsystem does not actually need the catalog information about referential relationships because it does not update the data.

Create Indexes

If you are using DB2 storage groups, define the storage group for both the owner and the readers with the same integrated catalog facility catalog name, even though the data sets are defined only in the DB2 subsystem containing the owner.

Create Indexes for the Owner

1. Start the database and table space related to the index for read-write access.
2. If you are managing your own data sets, define all data sets for the owner's shared indexes with VSAM SHAREOPTIONS(1,3), which allows either multiple readers *or* a single updater. If you let DB2 create data sets, this is taken care of automatically.
3. Issue CREATE INDEX in the owning DB2 subsystem, as shown in the following example:

```
CREATE UNIQUE INDEX EMPIX1
       ON DON.EMP
          (EMPID ASC)
       USING VCAT DONSCAT
       CLUSTER
       CLOSE NO
       BUFFERPOOL BP1;
```

   Hint: Create the index name with eight or fewer characters, and make sure that the name you choose does not exist already in the database for any other table spaces or indexes. By doing this, you can avoid having DB2 randomly generate the index space name part of the VSAM data set name.

Create Indexes for the Readers

1. Stop the index space in the owner.
2. If you didn't specify an index space name for the owner, you must determine the DB2-generated name. To determine the name used by DB2, use the following query against the owning DB2's catalog:

```
SELECT INDEXSPACE FROM SYSIBM.SYSINDEXES
  WHERE CREATOR = 'index_creator'
  AND NAME = 'index_name';
```

3. Issue your CREATE INDEX statement in the reading subsystem, specifying the same values as in the owner, except that you can specify a different value for BUFFERPOOL and CLOSE. PCTFREE and FREEPAGE are used when reorganizing an index and thus have no meaning for a reader.

Load Data in the Owner

Use the LOAD utility or SQL INSERT statements to put records into the owner's tables. For more information about the LOAD utility, see “Chapter 2-10. Loading Data into DB2 Tables” on page 2-113.

Run COPY and, optionally, RUNSTATS on the data. The COPY utility prepares you for recovery and the RUNSTATS utility updates catalog statistics. COPY prepares you for recovery. See “Appendix G. Using Tools to Monitor Performance” on page X-173 for information about RUNSTATS, and *Utility Guide and Reference* for information about COPY.

Starting and Stopping a Shared Database

This section discusses special considerations and recommendations for starting and stopping shared databases. As shown in Figure 179, an owner, DBX in this case, can be started for read-only access from the owning DB2 (DB2A) and can be read concurrently by the reading DB2s (DB2B and DB2C).

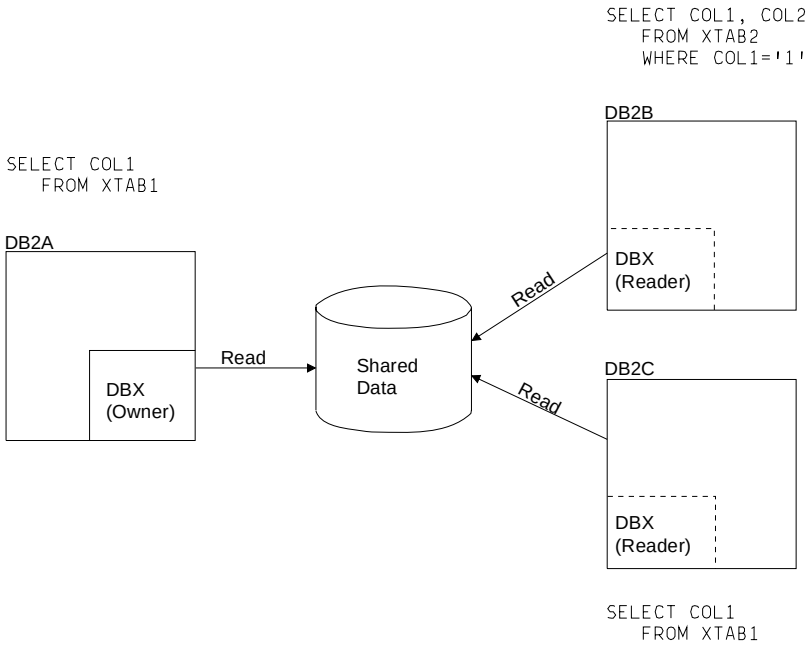


Figure 179. DB2s Reading Shared Data Concurrently. Any or all of the sharing databases, including the owner, can read the shared data, except when the owner is being updated. data to be read.

As Figure 180 on page X-165 illustrates, to update DBX, you must first stop any relevant index spaces and table spaces in all readers. Though you only need to stop index spaces, table spaces or partitions that need to be accessed, its easier to stop the entire database. Use the STOP DATABASE command with the SPACENAM option. Then, start the objects for read-write access in the owner.

Because no other DB2 can access shared data while it is being updated in the owner, you should schedule updates for a specified window of time.

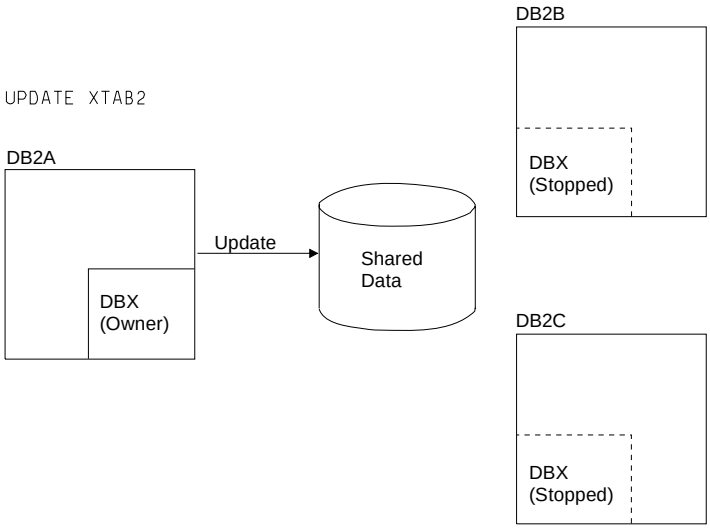


Figure 180. The Owning DB2 Updates Shared Data. The reading DB2s must stop access to the shared data while the owner is being updated.

When DB2 is through updating the data in the owner, you must stop access to the data by stopping the table spaces and index spaces. Then, the database can be restarted for read-only shared access by all sharing subsystems that read the data.

A single DB2 can have both owners and readers. For example, DB2A's database DBX can be the owner, and, at the same time, DB2A's DBY can be a reader. This configuration is shown in Figure 181, where you can see that DBY is defined as an owner in DB2B and DBX is defined as a reader. It also shows that shared data cannot be read and updated at the same time.

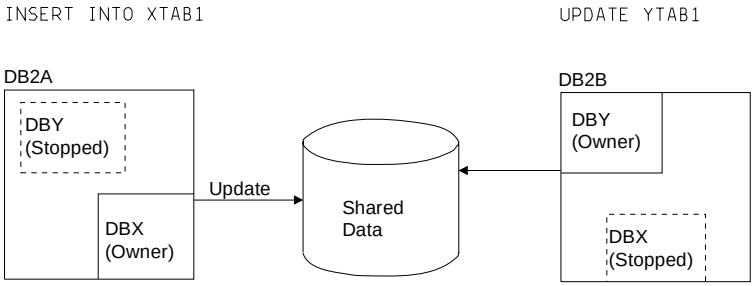


Figure 181. A Single DB2 Can Contain a Reader and an Owner. Because read and update activity cannot occur concurrently on a single shared database, the reader is stopped while the owner is updated.

Starting and stopping table spaces or partitions and index spaces in a shared database is an essential part of managing a shared database. For example, to update data in a specified table space or partition or index space, you have to stop the readers from accessing that object. If you do not, the object you want to update is not available for update access. Indeed, whenever you switch access modes in the owner (from read-write to read-only, or from read-only to read-write), you have to first stop the database (using the SPACENAM option) in the owner.

We recommend that you use START DATABASE and STOP DATABASE commands on the index space, rather than on the logical partition. For example, use:

```
START DB(dbname) SPACE (npi)
```

instead of

```
START DB(dbname) SPACE(npi) PART(n)
```

Also, if you switch from read-only to read-write, you must stop *all* shared databases, including the owner. If a table space or partition or index in the owner is started for read-write access, applications on the owner's DB2 cannot have any access to the data until all the readers have been stopped.

Starting a Shared Database

All the access options in the START DATABASE (*dbname*) SPACENAM (*) command (such as RO, RW, UT, FORCE) can be used for a shared database as well. These options are described in detail in Chapter 2 of *Command Reference* . Some of these options can have a different effect on shared databases:

- If you start table spaces or partitions and index spaces in a read-only database with ACCESS(RW) (the default), DISPLAY DATABASE shows that it is in read-write mode even though only read access is allowed. This is not an error condition; DB2 recognizes this as a read-only system and acts accordingly.
- You can enter a START DATABASE command with ACCESS(UT) on table spaces or partitions and index spaces in a reader, but this still allows only read access. (DISPLAY DATABASE shows the database started for UT access.) Thus, the only utilities that can work are those that do not actually update data. See “Running Utilities” on page X-170 for more information about running utilities on shared databases.
- Although we generally do not recommend it, there might be times when you need to enter a START DATABASE command with ACCESS(FORCE) on an owner's table spaces and index spaces to reset conditions that made the database unavailable. This puts the database into read-write mode. If you just want to read data, we recommend that you do the following after you start the database with ACCESS(FORCE):

  1. Enter the following command:

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

  2. Enter the following command:

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

Starting a Database for Update: Before updating an owner, stop all readers by entering the following command for all the reading DB2s:

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

Although it is possible to start the owner for read-write access before stopping the readers, you must stop the readers before updating because VSAM SHAREOPTIONS (1,3) allows only one updater *or* multiple readers. If you do not stop the readers, update requests in the owner fail with a “resource unavailable” reason code.

Starting a Database for Read-Only Access: When you want other systems to have read-only access to the data, enter the following command for the owner:

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

This quiesces all update activity. Then you can start the database spaces for read-only access for both the owner and the readers.

When you want to start a reader, we recommend that you explicitly specify ACCESS(RO). Even though a START DATABASE command with ACCESS(RW) is allowed for the reader, applications on the reader's DB2 are not allowed to update the data. Update requests for a read-only database fail with a “resource unavailable” reason code regardless of access mode.

Stopping a Shared Database

You can enter -STOP DATABASE (*dbname*) SPACENAM (*) from batch or from the console. We recommend that you enter the command from batch. When you enter the command from batch, you can submit a batch job stream that notifies someone to take action if the STOP DATABASE command cannot succeed. The request is handled synchronously, and DB2 waits for all activity to finish before returning control to the requester. If there is data inconsistency with the owner database, such as a permanent I/O error, the return code is set to 8.

When you enter -STOP DATABASE (*dbname*) SPACENAM (*) from the console, the request is handled asynchronously, and control returns to the operator before the database has stopped. For this reason, use a DISPLAY DATABASE command to ensure that all spaces are really stopped.

Maintaining Shared Read-Only Data

It is more complex to maintain and operate a shared database than a non-shared database. When planning a maintenance strategy for a shared read-only data system, consider the following:

- Rebinding plans and packages that use shared data

  Any time a change necessitates a rebind of a plan or package, rebind any equivalent plans or packages on the sharing DB2s. Do not share a PLAN_TABLE with a reader, because BIND with EXPLAIN(YES) fails when DB2 attempts to write to the table.
- “Updating,” below
- “Adding” on page X-168
- “Dropping” on page X-168
- “Altering” on page X-169
- “Running Utilities” on page X-170
- “Recovering” on page X-171

Updating

An application on the owner's DB2 cannot change the data until all sharing DB2s have stopped reading the shared data. This means you must stop the relevant table spaces or partitions and index spaces in all sharing DB2s. For this reason, we recommend that you schedule the owner's updates to occur in a specified time window.

Use the following steps to update the shared database:

1. Stop access to the shared databases by entering the following command for all shared databases, including the owner:

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

If the data sets are being accessed by non-DB2 facilities, such as access method services, then you must wait for those jobs to end before you can begin updating the database in the owner.

2. Enable the owner to be updated by entering the following command:

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

3. Update the data in the owner.
4. If the changes are significant, you could make an image copy and run RUNSTATS again on the owner. Also, consider rebinding plans and packages.
5. Enter the following command for the owner:

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

6. Allow read-only access by entering the following command for all shared databases, including the owner:

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

7. If you ran RUNSTATS on the owner's DB2 in step 4, now would be a good time to run RUNSTATS on the readers' DB2s. You might also consider rebinding plans and packages.

Adding

Before you add an object to a shared database, you must stop the database's table spaces and index spaces. The following procedure can be used to add an object to an existing shared database:

1. For all shared databases (including the owner), enter the following command:

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

2. Start the database spaces for read-write access by entering the following command for the owner:

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

3. Create the objects in the owner.
4. Stop the index spaces and table spaces in the owner by entering the following command:

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

5. Start the readers for read-only access by entering the following command:

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

6. Re-create the objects in the readers.

Dropping

To drop a shared database, or any object in the shared database:

1. Make sure no users are accessing the database by entering the following command for all shared databases, including the owner:

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

2. Drop the object from all the readers.
3. Drop the object from the owner.

If you do not follow the above steps in the given order, you can have unpredictable results because of the lack of synchronization between DB2 and VSAM.

Altering

With the ALTER DATABASE statement, you can change the default buffer pool, DB2 storage group, and shared status of the database. In this section, we describe how you change the shared status of a database, and some considerations for adding a column to an existing shared table.

Altering the Sharing Status of a Database

If you have SYSADM or SYSCTRL authority, you can convert the status of a database from non-shared to owner, or from owner to non-shared. However, when you have created a reader database, you cannot use SQL to convert its shared status. Instead, you must drop the database and redefine it. For information about altering or dropping a database, see “Altering DB2 Databases” on page 2-125.

Converting a Non-Shared Database to Sharing Owner: Follow the instructions in “Alter an Existing Database to be Shared” on page X-157.

Converting an Owner to Non-Shared:

To convert a database from sharing owner to non-shared:

1. Stop the database in all sharing DB2s (including the owning DB2) by entering the following command:

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

2. Drop this database from the reading subsystems. See “Dropping and Re-creating DB2 Objects” on page 2-123 for information about what happens when you drop a database.
3. For user-managed data sets, you can alter the VSAM SHAREOPTIONS from (1,3) to (3,3). DB2 does this for you with DB2-managed data sets.
4. Start the owner for read-write access by entering the following command:

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

5. Enter the ALTER DATABASE statement, specifying ROSHARE NONE:

   ```
   ALTER DATABASE DONSDB
     ROSHARE NONE;
   ```

6. Enter the following command for the owner:

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

   This makes the VSAM SHAREOPTIONS effective on the owner's storage group's data sets.

Adding a Column to a Table

For general information about altering tables, including information about adding columns to a table, see “Altering Tables” on page 2-128. Generally, the procedure is like that for adding objects to a database described in “Adding” on page X-168.

In the following procedure, we specify just the table space name on the -STOP command, but you can stop all the spaces in the database if you want.

1. For all shared databases (including the owner), enter the following command:

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

2. Start the table space for read-write access by entering the following command for the owner:

   ```
   -START DATABASE (dbname) SPACENAM (tsname) ACCESS(RW)
   ```

3. Use ALTER TABLE to add the column to the table in the owner. See Chapter 6 of *SQL Reference* for more information about the ALTER TABLE statement.
4. Stop the table space on the owner by entering the following command:

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

5. Start the readers for read-only access by entering the following command:

   ```
   -START DATABASE (dbname) SPACENAM (tsname) ACCESS(RO)
   ```

6. Alter the table in the readers.

DB2 does allow you to alter the table in the reader before altering it in the owner, but this is not recommended. You cannot add values to the column from the reader, and the column cannot be updated until it is added on the owner. If a column exists in the reader and not in the owner, a SELECT * from the reader returns that column. This can lead people to assume the column exists in the owner when it does not.

Running Utilities

All utilities that you can normally run on a database can be run on an owner. However, because the following utilities update data, you need to take special steps before running them:

RECOVER	COPY
REORG	REPAIR (DELETE and REPLACE)
LOAD	CHECK (DATA with DELETE option)

Use the following steps to run the above utilities:

1. For all shared databases (including the owner), enter the following command:

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

2. Enable utilities to be run on the owner by entering the following command:

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

3. Run the utility on the owner.
4. Enter the following command for the owner:

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

5. Allow read-only access by entering the following command on all shared databases, including the owner:

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

Because you cannot update a reader's data, the only utilities you can run on a reader are:

DIAGNOSE	DSN1COPY	RUNSTATS
DSN1COMP	DSN1PRNT	STOSPACE
DSN1CHKR	REPAIR DBD	

A note about STOSPACE: You can run the STOSPACE utility on both owners and readers because it makes the distinction between owner and reader data sets. If a given table space or index space is owned by the issuing system, then STOSPACE performs as it normally does for non-shared data sets. However, if it determines that the space is not owned by the issuing system, then it issues a message stating that fact and continues processing on the next table space or index space.

Recovering

When you perform a recovery, it is for the owner. If you use the DB2 RECOVER utility, then there are no changes for recovering shared data. However, if you use DB2's DSN1COPY utility or non-DB2 facilities to recover the database, then there are some additional steps:

DSN1COPY Utility

After you use DSN1COPY to restore some version of data:

1. Enter the following command:

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

2. Enter the following command:

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

After these steps, the data is in a consistent state, and you can start the database in any mode you want. If you do not follow the above steps, the data is considered inconsistent and is unavailable for access.

Non-DB2 Facilities

If you use non-DB2 facilities for data recovery (such as volume dump and restore), dump the data sets *after* you alter a database's shared status. This prevents recovering a data set to a state inconsistent with the current shared status of the database. This could occur when you recover to a time before the database was converted to shared.

However, if you do get a consistency error, the header page must be reformatted to reset the status indicator, and the VSAM SHAREOPTIONS must be made consistent with the shared status of the database. To do this:

1. Stop all shared databases, (including the owner), by entering the following command:

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

2. Start the owner for utility access by entering the following command:

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

3. Run REPAIR REPLACE RESET on table spaces and indexes in the owner:

   - If the spaces are not partitioned, reset only the header page.
   - If spaces are partitioned, reset the header page of each partition.

   For more information about the REPAIR utility, see Section 1 of *Utility Guide and Reference*.

4. Enter the following command for the owner:

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

5. Use the access method services LISTCAT command to check the VSAM SHAREOPTIONS of the data sets. Ensure that the SHAREOPTIONS of the recovered data sets are consistent with the ROSHARE attribute of the database:

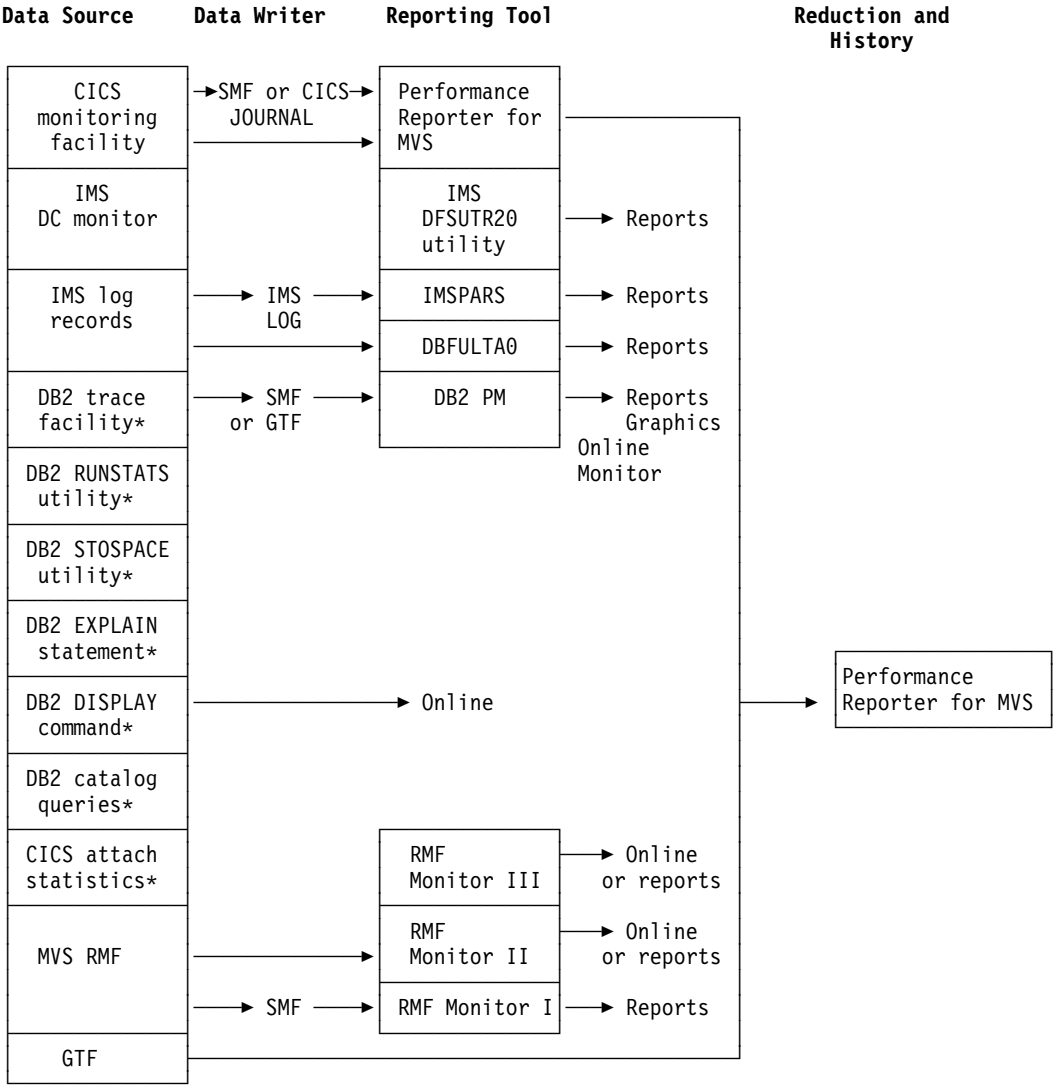
   If the database is ROSHARE OWNER, then the VSAM SHAREOPTIONS should be (1,3).
   If the database is not shared, then the VSAM SHAREOPTIONS should be (3,3).

   If the SHAREOPTIONS are inconsistent, use the access method services ALTER command with the SHAREOPTIONS option to change the values.

After you have done the above steps, the data is in a consistent state, and you can start the database in any mode you want.

Appendix G. 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 182 shows various monitoring tools that can be used in a DB2 environment.



* Facilities available within the DB2 product

Figure 182. 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 DC monitor, part of the IMS product, can be used to monitor DB2 processor time in IMS transactions.

IMS Performance Analysis and Reporting System (IMSPARS), 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 X-177.

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 X-182 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 X-183 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 X-184.

DB2 RUNSTATS utility can report space use and access path statistics in the DB2 catalog. See "Using RUNSTATS to Monitor and Update Statistics" on page 5-249 and Section 2 of *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 Section 2 of *Utility Guide and Reference* .

DB2 EXPLAIN statement provides information about the access paths used by DB2. See "Chapter 5-10. Using EXPLAIN to Improve SQL Performance" on page 5-261 and Chapter 6 of *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 5-59. For information about using the DISPLAY command to monitor distributed data activity, see "Using the DISPLAY Command" on page 5-321. For the detailed syntax of each command, refer to Chapter 2 of *Command Reference*.

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 X-184.

DB2 catalog queries help you determine when to reorganize table spaces and indexes. See the description of the REORG utility in Section 2 of *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 *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
- IMSPARS or its equivalent for response-time analysis
- IMS DC Monitor or its equivalent for 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 183 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 183. User-Created System Resources Report

The RMF reports used to produce the information in Figure 183 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:

- IMSPARS Management Exception Report
- IMS DC Monitoring
- 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 X-182, is General-use Programming Interface and Associated Guidance Information as defined in “Notices” on page xi.

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 another licensed program, IBM DATABASE 2 Performance Monitor (DB2 PM), 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 *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 X-107. 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 X-123 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 *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 X-182 and “Recording GTF Trace Data” on page X-183 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 you specified YES for SMF STATISTICS 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
- Number of commits and aborts
- The number of times certain SQL statements are issued
- Number of buffer pool requests
- Counts of certain locking events
- Processor resources consumed
- Thread wait times for various events
- RID pool processing
- Distributed processing
- 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. In CICS Version 4 and subsequent releases, 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 3-120.

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 3-6. Auditing Concerns” on page 3-119 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 *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 X-123.

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 5-43.

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))
```

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 171 on page X-109. For more information about SMF, refer to *MVS/ESA System Management Facilities (SMF)*.

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 X-184 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 X-107.

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

Glossary and Bibliography

Glossary

The following terms and abbreviations are defined as they are used in the DB2 library. If you do not find the term you are looking for, refer to the index or to *Dictionary of Computing*.

A

abend. Abnormal end of task.

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with DB2. A complete list of DB2 abend reason codes and their explanations is contained in *Messages and Codes*.

abnormal end of task (abend). Termination of a task, a job, or a subsystem because of an error condition that cannot be resolved during execution by recovery facilities.

access method services. A utility program that defines and manages VSAM data sets (or files).

access path. The path used to get to data specified in SQL statements. An access path can involve an index or a sequential search.

active log. The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records, whereas the archive log holds those records that are older and no longer will fit on the active log.

address space. A range of virtual storage pages identified by a number (ASID) and a collection of segment and page tables which map the virtual pages to real pages of the computer's memory.

address space connection. The result of connecting an allied address space to DB2. Each address space containing a task connected to DB2 has exactly one address space connection, even though more than one task control block (TCB) can be present. See *allied address space* and *task control block*.

alias. An alternate name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem.

allied address space. An area of storage external to DB2 that is connected to DB2 and is therefore capable of requesting DB2 services.

allied thread. A thread originating at the local DB2 subsystem that may access data at a remote DB2 subsystem.

already verified. An LU 6.2 security option which allows DB2 to provide the user's verified authorization ID when allocating a conversation. The user is not validated by the partner DB2.

ambiguous cursor. A database cursor that is not defined with either the clauses FOR FETCH ONLY or FOR UPDATE OF, is not defined on a read-only result table, is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement, and is in a plan or package that contains SQL statements PREPARE or EXECUTE IMMEDIATE.

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 used to define DB2 to VTAM as an application program using SNA LU 6.2 protocols.

application. A program or set of programs that perform a task; for example, a payroll application.

application plan. The control structure produced during the bind process and used by DB2 to process SQL statements encountered during statement execution.

application process. The unit to which resources and locks are allocated. An application process involves the execution of one or more programs.

application program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program.

application requester (AR). See *requester*.

application server. See *server*.

AR. application requester. See *requester*.

archive log. The portion of the DB2 log that contains log records that have been copied from the active log.

AS. Application server. See *server*.

ASCII. An encoding scheme used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC*.

attachment facility. An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

attribute. A characteristic of an entity. For example, in database design, the phone number of an employee is one of that employee's attributes.

authorization ID. A string that can be verified for connection to DB2 and to which a set of privileges are allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current release of a program.

authorized program facility (APF). A facility that permits the identification of programs that are authorized to use restricted functions.

B

backward log recovery. The fourth and final phase of restart processing during which DB2 scans the log in a backward direction to apply UNDO log records for all aborted changes.

base table. A table created by the SQL CREATE TABLE statement that is used to hold persistent data. Contrast with *result table* and *temporary table*.

basic sequential access method (BSAM). An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential access or a direct access device.

bind. The process by which the output from the DB2 precompiler is converted to a usable control structure called a package or an application plan. During the process, access paths to the data are selected and some authorization checking is performed.

> **automatic bind**. (More correctly *automatic rebind*). A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.
> **dynamic bind**. A process by which SQL statements are bound as they are entered.
> **incremental bind**. A process by which SQL statements are bound during the execution of an application process, because they could not be bound during the bind process, and VALIDATE(RUN) was specified.
> **static bind**. A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time. Contrast with *dynamic bind*.

BMP. Batch Message Processing (IMS).

bootstrap data set (BSDS). A VSAM data set that contains name and status information for DB2, as well as RBA range specifications, for all active and archive log data sets. It also contains passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.

BSAM. Basic sequential access method.

BSDS. Bootstrap data set.

buffer pool. Main storage reserved to satisfy the buffering requirements for one or more table spaces or indexes.

built-in function. Scalar function or column function.

C

CAF. Call attachment facility.

call attachment facility (CAF). A DB2 attachment facility for application programs running in TSO or MVS batch. The CAF is an alternative to the DSN command processor and allows greater control over the execution environment.

cascade delete. The enforcement of referential constraints by DB2 when it deletes all descendent rows of a deleted parent row.

catalog. In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

catalog table. Any table in the DB2 catalog.

CCSID. Coded character set identifier.

CDB. See *communications database*.

CEC. 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 set. A defined set of characters.

character string. A sequence of bytes representing bit data, single-byte characters, or a mixture of single and double-byte characters.

check clause. An extension to the SQL CREATE TABLE and SQL ALTER TABLE statements that specifies a table check constraint.

check constraint. See *table check constraint*.

check integrity. The condition that exists when each row in a table conforms to the table check constraints defined on that table. Maintaining check integrity requires enforcing table check constraints on operations that add or change data.

check pending. A state of a table space or partition that prevents its use by some utilities and some SQL statements, because it can contain rows that violate referential constraints, table check constraints, or both.

checkpoint. A point at which DB2 records internal status information on the DB2 log that would be used in the recovery process if DB2 should abend.

CI. Control interval.

CICS. Represents (in this publication) CICS/MVS and CICS/ESA.

> **CICS/MVS**: Customer Information Control System/Multiple Virtual Storage.
> **CICS/ESA**: Customer Information Control System/Enterprise Systems Architecture.

CICS attachment facility. A DB2 subcomponent that uses the MVS Subsystem Interface (SSI) and cross storage linkage to process requests from CICS to DB2 and to coordinate resource commitment.

CIDF. Control interval definition field.

claim. To register to DB2 that an object is being accessed. This registration is also called a claim. A claim is used to ensure that an object cannot be drained until a commit is reached. Contrast with *drain*.

claim class. A specific type of object access which can be one of the following:

> cursor stability (CS)
> repeatable read (RR)
> write

claim count. A count of the number of agents that are accessing an object.

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.

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 SQL operation that derives its result from a collection of values across one or more rows. Contrast with *scalar function*.

"come from" checking. An LU 6.2 security option which 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. Distinct from an SQL statement.

command recognition character (CRC). A character that permits an MVS console operator or an IMS subsystem user to route DB2 commands to specific DB2 subsystems.

commit. The operation that ends a unit of work by releasing locks so that the database changes made by that unit of work can be perceived by other processes.

commit point. A point in time when data is considered consistent.

committed phase. The second phase of the multi-site update process that requests all participants to commit the effects of the logical unit of work.

common service area (CSA). In MVS, a part of the common area that contains data areas 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 =, >, <) 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 supplied by the attachment facility that is associated with a specific address space connection.

consistency token. A timestamp used to generate the version identifier for an application. See also *version*.

constraint. A rule that limits the values that can be inserted, deleted, or updated in a table. See *referential constraint*, *uniqueness constraint*, and *table check constraint*.

control interval (CI). A fixed-length area or direct access storage in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records pointed to by an entry in the sequence-set index record. The control interval is the unit of information that VSAM transmits to or from direct access storage. A control interval always includes an integral number of physical records.

control interval definition field (CIDF). In VSAM, a field located in the four bytes at the end of each control interval; it describes the free space, if any, in the control interval.

conversation. (1) A VTAM term for a dialog between two application processes, on different DB2 subsystems, that is specified by a particular *session name*, *mode name*, and *LU name*. (2) An LU 6.2 security option which allows DB2 to require the user's authorization ID and password when allocating a conversation to a partner DB2. The user is validated by the partner DB2.

coordinator. The system component that coordinates the commit or rollback of a unit of work that includes work done on one or more other systems.

correlated subquery. A subquery (part of a WHERE or HAVING clause) applied to a row or group of rows of a table or view named in an outer sub-SELECT statement.

correlation ID. An identifier 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.

CPC. See *central processor complex*.

CRC. Command recognition character.

CRCR. Conditional restart control record.

cross-memory linkage. A method for invoking a program in a different address space. The invocation is synchronous with respect to the caller.

CSA. Common service area.

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 used by an application program 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 cursor table is the copy of the skeleton cursor table used by an executing application process.

cycle. A set of tables that can be ordered so that each table is a descendent of the one before it, and the first is a descendent of the last. A self-referencing table is a cycle with a single member.

D

DASD. Direct access storage device.

database. A collection of tables, or a collection of table spaces and index spaces.

database access thread. A thread accessing data at the local subsystem on behalf of a remote subsystem.

database administrator (DBA). An individual responsible for the design, development, operation, safeguarding, maintenance, and use of a database.

database descriptor (DBD). An internal representation of DB2 database definition which reflects the data definition found in the DB2 catalog. The objects defined in a database descriptor are table spaces, tables, indexes, index spaces, and relationships.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

database request module (DBRM). A data set member created by the DB2 precompiler that contains information about SQL statements. DBRMs are used in the bind process.

DATABASE 2 Interactive (DB2I). The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands, and utility invocation.

data definition name (DD name). The name of a data definition (DD) statement that corresponds to a data control block containing the same name.

Data Language/I (DL/I). The IMS data manipulation language; a common high-level interface between a user application and IMS.

data 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 assigned by XCF services to a data sharing group.

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.

DBCS. Double-byte character set.

DBD. Database descriptor.

DBID. Database identifier.

DBMS. Database management system.

DBRM. Database request module.

DB2 catalog. Tables maintained by DB2 that contain descriptions of DB2 objects such as tables, views, and indexes.

DB2 command. An instruction to the DB2 subsystem allowing a user to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases, and so on.

DB2I. DATABASE 2 Interactive.

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.

DB2 private protocol access. A method of accessing distributed data by which you can direct a query to another DB2 system by using an alias or a three-part name to identify the DB2 subsystems at which the statements are executed. Contrast with *DRDA access*.

DB2 private protocol connection. A DB2 private connection of the application process. See also *private connection*.

DCLGEN. Declarations generator.

DDF. Distributed data facility.

DD name. Data definition name.

deadlock. Unresolvable contention for the use of a resource such as a table or an index.

declarations generator (DCLGEN). A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated

from DB2 system catalog information. DCLGEN is also a DSN subcommand.

default value. A predetermined value, attribute, or option that is assumed when no other is explicitly specified.

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) is a dependent if it has at least one parent. The object is also said to be a dependent (row, table, or table space) of its parent. See *parent row*, *parent table*, *parent table space*.

dependent row. A row that contains a foreign key that matches the value of a primary key in the parent row.

dependent table. A table that is dependent in at least one referential constraint.

descendent. An object is a descendent of another object if it is a dependent of the object, or if it is the dependent of a descendent of that object.

descendent row. A row that is dependent on another row or a row that is a dependent of a descendent row.

descendent table. A table that is a dependent of another table or a dependent of a descendent table.

DFHSM. Data Facility Hierarchical Storage Manager.

DFP. Data Facility Product (MVS).

direct access storage device (DASD). A device in which access time is independent of the location of the data.

directory. The system database that contains internal objects such as database descriptors and skeleton cursor tables.

distributed data facility (DDF). A set of DB2 components through which DB2 communicates with another RDBMS.

distributed relational database architecture (DRDA). A connection protocol for distributed relational database processing that is used by IBM's relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems.

DL/I. Data Language/I. The IMS data manipulation language; a common high-level interface between a user application and IMS.

double-byte character set (DBCS). A set of characters used by national languages such as Japanese and Chinese that have more symbols than can be represented by a single byte. Each character is two bytes in length, and therefore requires special hardware to be displayed or printed.

drain. To acquire a locked resource by quiescing access to that object.

drain lock. A lock on a claim class which prevents a claim from occurring.

DRDA. Distributed relational database architecture.

DRDA access. A method of accessing distributed data by which you can explicitly connect to another location, using an SQL statement, to execute packages that have been previously bound at that location. The SQL CONNECT statement is used to identify application servers, and SQL statements are executed using packages that were previously bound at those servers. Contrast with *DB2 private protocol access*.

DSN. (1) The default DB2 subsystem name. (2) The name of the TSO command processor of DB2. (3) The first three characters of DB2 module and macro names.

duration. A number that represents an interval of time. See *date duration*, *labeled duration*, and *time duration*.

dynamic SQL. SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution.

E

EBCDIC. Extended binary coded decimal interchange code. An encoding scheme used to represent character data in the MVS, VM, VSE, and OS/400 environments. Contrast with *ASCII*.

EDM pool. A pool of main storage used for database descriptors and application plans.

EID. Event identifier.

embedded SQL. SQL statements coded within an application program. See *static SQL*.

EOM. End of memory.

EOT. End of task.

error page range. Range of pages considered to be physically damaged. DB2 will not allow a user to access any pages that fall within this range.

equi-join. A join operation in which the join-condition has the form *expression* = *expression*.

ESDS. Entry sequenced data set.

ESMT. External subsystem module table (IMS).

EUR. IBM European Standards.

exception table. A table that holds rows that violate referential constraints or table check constraints found by the CHECK DATA utility.

exclusive lock. A lock that prevents concurrently executing application processes from reading or changing data. Contrast with *shared lock*.

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.

F

fallback. The process of returning to a previous release of DB2 after attempting or completing migration to a current release.

field procedure. A user-written exit routine designed to receive a single value and transform (encode or decode) it in any way the user can specify.

fixed-length string. A character or graphic string whose length is specified and cannot be changed. Contrast with *varying-length string*.

foreign key. A key that is specified in the definition of a referential constraint. Because of the foreign key, the table is a dependent table. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

forward log recovery. The third phase of restart processing during which DB2 processes the log in a forward direction to apply all REDO log records.

free space. The total unused space in a page, that is, the space not used to store records or control information.

full outer join. The result of a join operation that includes the matched rows of both tables being joined and preserves the unmatched rows of both tables. See also *join*.

function. A scalar function or column function. Same as *built-in function*.

G

GB. Gigabyte (1,073,741,824 bytes).

generalized trace facility (GTF). An MVS service program that records significant system events such as I/O interrupts, SVC interrupts, program interrupts, or external interrupts.

generic resource name. A name used by VTAM that represents several application programs that provide the same function in order to handle session distribution and balancing in a Sysplex.

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. A coupling facility cache structure used by a data sharing group to cache data and to ensure that the data is consistent for all members.

GTF. Generalized trace facility.

H

help panel. A screen of information presenting tutorial text to assist a user at the terminal.

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 written in a host language that contains embedded SQL statements.

host structure. In an application program, a structure referenced by embedded SQL statements.

host variable. In an application program, an application variable referenced by embedded SQL statements.

HSM. Hierarchical storage manager.

I

ICF. Integrated catalog facility.

IDCAMS. An IBM program used to process access method services (AMS) commands. It can be invoked as a job or jobstep, from a TSO terminal, or from within a user's application program.

IDCAMS LISTCAT. A facility for obtaining information contained in the access method services catalog.

identify. A request that an attachment service program in an address space separate from DB2 issues via the MVS subsystem interface to inform DB2 of its existence and initiate the process of becoming connected to DB2.

IFCID. Instrumentation facility component identifier.

IFI. Instrumentation facility interface.

IFI call. An invocation of the instrumentation facility interface (IFI) by means of one of its defined functions.

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 will continue to back out the changes during restart.

in-commit. A status of a unit of recovery. If DB2 fails after beginning its phase 2 commit processing, it “knows,” when restarted, that changes made to data are consistent. Such units of recovery are termed *in-commit*.

independent. An object (row, table, or table space) is independent if it 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 used to determine the order of index entries.

index partition. A VSAM data set that is contained within a partitioned index space.

index space. A page set used to store the entries of one index.

indicator variable. A variable 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 this unit of recovery is to be committed or rolled back. At emergency restart, if DB2 does not have the information needed to make this decision, its unit of recovery is *indoubt* until DB2 obtains this information from the coordinator.

indoubt resolution. The process of resolving the status of an indoubt logical unit of work to either the committed or the rollback state.

inflight. A status of a unit of recovery. If DB2 fails before its unit of recovery completes phase 1 of the commit process, it merely backs out the updates of its unit of recovery when it is restarted. These units of recovery are termed *inflight*.

inner join. The result of a join operation that includes only the matched rows of both tables being joined. See also *join*.

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). Names a traceable event and identifies the trace record of that event. As a parameter on the -START TRACE and -MODIFY TRACE commands, it specifies tracing the corresponding event.

Interactive System Productivity Facility (ISPF). An IBM licensed program that provides interactive dialog services.

internal resource lock manager (IRLM). An MVS subsystem used by DB2 to control communication and database locking.

IRLM. internal resource lock manager.

ISO. International Standards Organization.

isolation level. The degree to which a unit of work is isolated from the updating operations of other units of work. See also *cursor stability*, *repeatable read*, *uncommitted read*, and *read stability*.

ISPF. Interactive System Productivity Facility.

ISPF/PDF. Interactive System Productivity Facility/Program Development Facility.

J

JCL. Job control language.

JES. MVS Job Entry Subsystem.

JIS. Japanese Industrial Standard.

join. A relational operation that allows retrieval of data from two or more tables based on matching column values. See also *full outer join, inner join, left outer join, outer join, right outer join, equi-join*.

K

KB. Kilobyte (1024 bytes).

key. A column or an ordered collection of columns identified in the description of a table, index, or referential constraint.

KSDS. Key sequenced data set.

L

labeled duration. A number that represents a duration of years, months, days, hours, minutes, seconds, or microseconds.

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 being joined, and 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.

link-edit. To create a loadable computer program using a linkage editor.

L-lock. See *logical lock*.

load module. A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

local 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 or page lock to a table space lock because the number of page locks concurrently held on a given resource exceeds a preset limit.

locking. The process by which the integrity of data is ensured. Locking prevents concurrent users from accessing inconsistent data.

lock mode. A representation for the type of access concurrently running programs can have to a resource held by a DB2 lock.

lock object. The resource that is controlled by a DB2 lock.

lock promotion. The process of changing the size or mode of a DB2 lock to a higher level.

lock size. The amount of data controlled by a DB2 lock on table data; the value can be a row, a page, a table, or a table space.

log. A collection of records that describe the events that occur during DB2 execution and their sequence. The information thus recorded is used for recovery in the event of a failure during DB2 execution.

logical index partition. The set of all keys that reference the same data partition.

logical lock. The lock type used by transactions to control intra- and inter-DB2 data concurrency between transactions.

logical 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). In IMS, the processing that program performs between synchronization points.

logical unit of work identifier (LUWID). A name that uniquely identifies a thread within a network. This name consists of a fully-qualified LU network name, an LUW instance number, and an LUW sequence number.

log initialization. The first phase of restart processing during which DB2 attempts to locate the current end of the log.

log record sequence number (LRSN). A number DB2 generates and associates with each log record. DB2 also uses the LRSN for page versioning. The LRSNs generated by a given DB2 data sharing group form a strictly increasing sequence for each DB2 log and a strictly increasing sequence for each page across the DB2 group.

log truncation. A process by which an explicit starting RBA is established. This RBA is the point at which the next byte of log data will be written.

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.

LRECP. Logical recovery pending.

LRH. Log record header.

LRSN. See *log record sequence number*.

LUW. Logical unit of work.

LUWID. Logical unit of work identifier.

M

materialize. The process of putting rows from a view or nested table expression into a work file for further processing by a query.

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 which 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 (IMS).

MSS. Mass Storage Subsystem

MTO. Master terminal operator.

multi-site update. Distributed relational database processing in which data is updated in more than one location within a single unit of work.

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.

MVS/XA. Multiple Virtual Storage/Extended Architecture.

N

nested table expression. A subselect in a FROM clause (surrounded by parentheses).

NID (network identifier). The network ID assigned by IMS or CICS, or if the connection type is RRSAF, the OS/390 RRS Unit of Recovery ID (URID).

nonleaf page. A page that contains keys and page numbers of other pages in the index (either leaf or nonleaf pages). Nonleaf pages never point to actual data.

nonpartitioned index. Any index that is not a partitioned 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 assigned sequentially to each IMS schedule since the last cold start of IMS and used as an identifier for a unit of work. In an 8-byte format, the first four bytes contain the schedule number and the last four contain the number of IMS sync points (*commit points*) during the current schedule. The OASN is part of the NID for an IMS connection.

OBID. Data object identifier.

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.

outer join. The result of a join operation that includes the matched rows of both tables being joined and preserves some or all of the unmatched rows of the tables being joined. See also *join*.

P

package. Also *application package*. An object containing a set of SQL statements that have been bound statically and that are available for processing.

package list. An ordered list of package names that may be used to extend an application plan.

package name. The name given an object created by a BIND PACKAGE or REBIND PACKAGE command. The object is a bound version of a database request module (DBRM). The name consists of a location name, a collection ID, a package ID, and a version ID.

page. A unit of storage within a table space (4KB or 32KB) or index space (4KB). In a table space, a page contains one or more rows of a table.

page set. A table space or index space consisting of pages that are either 4KB or 32KB in size. Each page set is made from a collection of VSAM data sets.

page set recovery pending (PSRCP). A restrictive state of an index space in which the page set is in a recovery pending state. In this case, the entire page set must be recovered. Recovery of a logical part is prohibited.

parallel group. A set of consecutive operations executed in parallel 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 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. Synonymous with *agent* in SNA.

partition. A portion of a page set. Each partition corresponds to a single, independently extendable data set. Partitions can be extended to a maximum size of 1, 2, or 4 gigabytes, depending upon the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. 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 subdivided into parts (based upon index key range), each of which may be processed by utilities independently.

partner logical unit. An access point in the SNA network that is connected to the local DB2 by way of a VTAM conversation.

PCT. Program control table (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 to execute it.

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 (CICS).

point of consistency. A time when all recoverable data an application accesses is consistent with other data. Synonymous with *sync point* or *commit point*.

PPT. (1) Processing program table (CICS). (2) Program properties table (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. 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.

private connection. A communications connection that is specific to DB2.

privilege. The capability of performing a specific function, sometimes on a specific object. The term includes:

- **explicit privileges**, which have names and are held as the result of SQL GRANT and REVOKE statements. For example, the SELECT privilege.
- **implicit privileges**, which accompany the ownership of an object, such as the privilege to drop a synonym one owns, or the holding of an authority, such as the privilege of SYSADM authority to terminate any utility job.

privilege set. For the installation SYSADM ID, the set of all possible privileges. For any other authorization ID, the set of all privileges recorded for that ID in the DB2 catalog.

process. A general term for a unit that depends on the environment, but has the same basic properties in every environment. A process involves the execution of one or more programs, and is the unit to which resources and locks are allocated. The execution of an SQL statement is always associated with some process.

program. A single compilable collection of executable statements in a programming language.

program temporary fix (PTF). A solution or bypass of a problem diagnosed as a result of a defect in a current unaltered release of a licensed program. An authorized program analysis report (APAR) fix is corrective service for an existing problem. A PTF is preventive service for problems that might be encountered by other users of the product. A PTF is “temporary” because a permanent fix is usually not incorporated into the product until its next release.

protected conversation. A VTAM conversation that supports two-phase commit flows.

PSRCP. Page set recovery pending.

PTF. Program temporary fix.

Q

QMF. Query Management Facility.

QSAM. Queued Sequential Access Method.

query CP parallelism. Parallel execution of a single query accomplished by using multiple tasks. See also *Sysplex query parallelism*.

query I/O parallelism. Parallel access of data accomplished by triggering multiple I/O requests within a single query.

R

RACF. OS/VS2 MVS Resource Access Control Facility.

RAMAC. IBM family of enterprise disk storage system products.

RBA. Relative byte address.

RCT. Resource control table (CICS attachment facility).

RDB. See *relational database*.

RDBMS. Relational database management system.

RDBNAM. See *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, known as *phantom rows*, that were inserted and committed by a concurrently executing application process.

rebind. To create a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table accessed by your application, you must rebind the application in order to take advantage of that index.

record. The storage representation of a row or other data.

record identifier (RID) pool. An area of main storage above the 16MB line that is reserved for sorting record identifiers during list prefetch processing.

recovery. The process of rebuilding databases after a system failure.

recovery log. A collection of records that describes the events that occur during DB2 execution and their sequence. The information recorded is used for recovery in the event of a failure during DB2 execution.

recovery pending (RECP). This condition prevents SQL access to a table space or index space that may need to be recovered.

recovery token. An identifier for an element used in recovery. For example, *NID* or *URID*.

RECP. Recovery pending.

redo. A state of a unit of recovery which indicates that changes made are to be reapplied to the DASD media to ensure data integrity.

referential constraint. The requirement that nonnull values of a designated foreign key are valid only if they equal values of the primary key of a designated table.

referential integrity. The condition that exists when all intended references from data in one column of a table to data in another column of the same or a different table are valid. Maintaining referential integrity requires enforcing referential constraints on all LOAD, RECOVER, INSERT, UPDATE, and DELETE operations.

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.

relational database. 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 relational database manager that operates consistently across supported IBM systems.

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 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 DB2subsystem. Specifically, the request 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 may even operate on the same processor under the same MVS system.

repeatable read (RR). The isolation level that provides maximum protection from other executing application programs. When an application program executes with repeatable read protection, rows referenced by the program cannot be changed by other programs until the program reaches a commit point.

request commit. The vote submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

requester. Also *application requester (AR)*. The source of a request to a remote RDBMS, the system that requests the data.

request unit (RU). The part of a basic information unit that follows a request header and contains the data.

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 allows you to define CICS resources on line without assembling tables.

resource limit facility (RLF). A portion of DB2 code that prevents dynamic manipulative SQL statements from exceeding specified time limits.

resource limit specification table. A site-defined table that specifies the limits to be enforced by the resource limit facility.

result table. The set of rows specified by a SELECT statement.

RID pool. Record identifier pool.

right outer join. The result of a join operation that includes the matched rows of both tables 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.

row. The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

row lock. A lock on a single row of data.

RRE. Residual recovery entry (IMS).

RRSAF. Recoverable Resource Manager Services attachment facility. 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.

RTT. Resource translation table.

RU. Request unit.

S

SBCS. Single-byte character set.

scalar function. An SQL operation that produces a single value from another value and is expressed as a function name followed by a list of arguments enclosed in parentheses. See also *column function*.

search condition. A criterion for selecting rows from a table. A search condition consists of one or more predicates.

secondary authorization ID. An authorization ID that has been associated with a primary authorization ID by an authorization exit routine.

section. The segment of a plan or package that contains the executable structures for a single SQL statement. For most SQL statements, there is one section in the plan for each SQL statement in the source program. However, for cursor-related statements, the DECLARE, OPEN, FETCH, and CLOSE reference the same section because they each refer to the SELECT statement 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. Also *application server (AS)*. The target for a request from a remote RDBMS, the RDBMS that provides the data.

session. A link between two nodes in a VTAM network.

session protocols. The available set of SNA communication requests and responses.

shared lock. A lock that prevents concurrently executing application processes from changing data, but not from reading data.

short string. A string whose actual length, or a varying-length string whose maximum length, is 255 bytes (127 double-byte characters) or less.

sign-on. A request made on behalf of an individual CICS or IMS application process by an attach facility to enable DB2 to verify that it is authorized to use DB2 resources.

simple 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 gigabytes, another data set is created, and so on up to a total of 32 data sets. The data sets are considered by DB2 to be a single contiguous linear address space containing a maximum of 64 gigabytes. 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.

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).

special register. A storage area that is defined for a process by DB2 and is used to store information that can be referenced in SQL statements. Examples of special registers are USER, CURRENT DATE, and CURRENT TIME.

SPUFI. SQL Processor Using File Input. A facility of the TSO attachment subcomponent that enables the DB2I user to execute SQL statements without embedding them in an application program.

SQL. Structured Query Language.

SQL authorization ID (SQL ID). The authorization ID that is used for checking dynamic SQL statements in some situations.

SQL communication area (SQLCA). A structure used to provide an application program with information about the execution of its SQL statements.

SQL descriptor area (SQLDA). A structure that describes input variables, output variables, or the columns of a result table.

SQL processing conversation. Any conversation that requires access of DB2 data, either through an application or by dynamic query requests.

SQLCA. SQL communication area.

SQLDA. SQL descriptor area.

SQL/DS. SQL/Data System. Also known as *DB2/VSE & VM*.

SSI. MVS subsystem interface.

SSM. Subsystem member.

stand-alone. An attribute of a program that means it is capable of executing separately from DB2, without using DB2 services.

static SQL. SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables specified by the statement might change).

storage group. A named set of DASD volumes on which DB2 data can be stored.

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 RDBMS.

sync point. See *commit point*.

synonym. In SQL, an alternative name for a table or view. Synonyms can only be used to refer to objects at the subsystem in which the synonym is defined.

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.

Sysplex query parallelism. Parallel execution of a single query accomplished by using multiple tasks on more than one DB2. See also *query CP parallelism*.

system administrator. The person having the second highest level of authority within DB2. System administrators make decisions about how DB2 is to be used and implement those decisions by choosing system parameters. They monitor the system and change its characteristics to meet changing requirements and new data processing goals.

system agent. A work request that DB2 creates internally.

system conversation. The conversation that two DB2s must establish to process system messages before any distributed processing can begin.

system diagnostic work area (SDWA). The data that is recorded in a SYS1.LOGREC entry that describes a program or hardware error.

System Modification Program/Extended (SMP/E). A tool for making software changes in programming systems (such as DB2 or MVS), 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. Synonymous with *base table* or *temporary table*.

table check constraint. A user-defined constraint that specifies the values that specific columns of a base table can contain.

table space. A page set used to store the records in one or more tables.

table space set. A set of table spaces and partitions that should be recovered together because each of them contains a table that is a parent or descendent of a table in one of the others.

task control block (TCB). A control block used to communicate information about tasks within an address space that are connected to DB2. An address space can support many task connections (as many as one per task), but only one address space connection. See *address space connection*.

TCB. MVS task control block.

temporary table. A table created by the SQL CREATE GLOBAL TEMPORARY TABLE statement that is used to hold temporary data. Contrast with *result table* and *temporary table*.

thread. The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure. See also *allied thread* and *database access thread*.

three-part name. The full name of a table, view, or alias. It consists of a location name, authorization ID, and an object name separated by a period.

time. A three-part value that designates a time of day in hours, minutes, and seconds.

time duration. A decimal integer that represents a number of hours, minutes, and seconds.

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 will wait for IRLM services after starting, and the amount of time IRLM will wait if a resource requested by an application is unavailable. If either of these time specifications is exceeded, a timeout is declared.

time-sharing option (TSO). Provides interactive time sharing from remote terminals.

timestamp. A seven-part value that consists of a date and time expressed in years, months, days, hours, minutes, seconds, and microseconds.

TMP. Terminal Monitor Program.

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 IMSenvironments can run under the TSO attachment facility.

type 1 indexes. Indexes that were created by a release of DB2 before DB2 Version 4 or that are specified as type 1 indexes in Version 4. Contrast with *type 2 indexes*.

type 2 indexes. A new type of indexes available in Version 4. They differ from *type 1 indexes* in several respects; for example, they are the only indexes allowed on a table space that uses *row locks*.

U

uncommitted read (UR). The isolation level that allows an application to read uncommitted data.

undo. A state of a unit of recovery that indicates that the changes made by the unit of recovery to recoverable DB2 resources must be backed out.

UNION. An SQL operation that combines the results of two select statements. UNION is often used to merge lists of values obtained from several tables.

unique index. An index which ensures that no identical key values are stored in a table.

uniqueness constraint. The rule that no two values in a primary key or key of a unique index can be the same.

unlock. To release an object or system resource that was previously locked and return it to general availability within DB2.

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.

UT. Utility-only access.

V

value. The smallest unit of data 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, or packages.

> **A version of a program** is the source code produced by precompiling the program. The program version is identified by the program name and a timestamp (consistency token).
> **A version of a DBRM** is the DBRM produced by precompiling a program. The DBRM version is identified by the same program name and timestamp as a corresponding program version.
> **A version of a package** is the result of binding a DBRM within a particular database system. The package version is identified by the same program name and consistency token as the DBRM.

view. An alternative representation of data from one or more tables. A view can include all or some of the columns contained in tables on which it is defined.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network.

VSAM. Virtual storage access method.

VTAM. MVS Virtual telecommunication access method.

W

WTO. Write to operator.

WTOR. Write to operator with reply.

X

XRF. Extended recovery facility.

Bibliography

DB2 for OS/390 Version 5

- *Administration Guide, SC26-8957*
- *Application Programming and SQL Guide, SC26-8958*
- *Call Level Interface Guide and Reference, SC26-8959*
- *Command Reference, SC26-8960*
- *Data Sharing: Planning and Administration, SC26-8961*
- *Data Sharing Quick Reference Card, SX26-3841*
- *Diagnosis Guide and Reference, LY27-9659*
- *Diagnostic Quick Reference Card, LY27-9660*
- *Installation Guide, GC26-8970*
- *Application Programming Guide and Reference for Java™, SC26-9547*
- *Licensed Program Specifications, GC26-8969*
- *Messages and Codes, GC26-8979*
- *Reference for Remote DRDA Requesters and Servers, SC26-8964*
- *Reference Summary, SX26-3842*
- *Release Guide, SC26-8965*
- *SQL Reference, SC26-8966*
- *Utility Guide and Reference, SC26-8967*
- *What's New?, GC26-8971*
- *Program Directory*

DB2 PM for OS/390 Version 5

- *Batch User's Guide, SC26-8991*
- *Command Reference, SC26-8987*
- *General Information, GC26-8982*
- *Getting Started on the Workstation, SC26-8989*
- *Master Index, SC26-8984*
- *Messages Manual, SC26-8988*
- *Online Monitor User's Guide, SC26-8990*
- *Report Reference Volume 1, SC26-8985*
- *Report Reference Volume 2, SC26-8986*
- *Program Directory*

Ada/370

- *IBM Ada/370 Language Reference, SC09-1297*
- *IBM Ada/370 Programmer's Guide, SC09-1414*
- *IBM Ada/370 SQL Module Processor for DB2 Database Manager User's Guide, SC09-1450*

APL2

- *APL2 Programming Guide, SH21-1072*
- *APL2 Programming: Language Reference, SH21-1061*
- *APL2 Programming: Using Structured Query Language (SQL), SH21-1057*

AS/400

- *DB2 for OS/400 SQL Programming, SC41-4611*
- *DB2 for OS/400 SQL Reference, SC41-4612*

BASIC

- *IBM BASIC/MVS Language Reference, GC26-4026*
- *IBM BASIC/MVS Programming Guide, SC26-4027*

C/370

- *IBM SAA AD/Cycle C/370 Programming Guide, SC09-1356*
- *IBM SAA AD/Cycle C/370 Programming Guide for Language Environment/370, SC09-1840*
- *IBM SAA AD/Cycle C/370 User's Guide, SC09-1763*
- *SAA CPI C Reference, SC09-1308*

Character Data Representation Architecture

- *Character Data Representation Architecture Overview, GC09-2207*
- *Character Data Representation Architecture Reference, SC09-2190*

CICS/ESA

- *CICS/ESA Application Programming Guide, SC33-1169*
- *CICS/ESA Application Programming Reference, SC33-1170*
- *CICS/ESA CICS - RACF Security Guide, SC33-1185*
- *CICS/ESA CICS-Supplied Transactions, SC33-1168*
- *CICS/ESA Customization Guide, SC33-1165*
- *CICS/ESA Data Areas, LY33-6083*
- *CICS/ESA Installation Guide, SC33-1163*
- *CICS/ESA Intercommunication Guide, SC33-1181*
- *CICS/ESA Messages and Codes, SC33-1177*
- *CICS/ESA Operations and Utilities Guide, SC33-1167*
- *CICS/ESA Performance Guide, SC33-1183*
- *CICS/ESA Problem Determination Guide, SC33-1176*
- *CICS/ESA Resource Definition Guide, SC33-1166*
- *CICS/ESA System Definition Guide, SC33-1164*
- *CICS/ESA System Programming Reference, GC33-1171*

CICS/MVS

- *CICS/MVS Application Programming Primer, SC33-0139*
- *CICS/MVS Application Programmer's Reference, SC33-0512*
- *CICS/MVS Facilities and Planning Guide, SC33-0504*
- *CICS/MVS Installation Guide, SC33-0506*
- *CICS/MVS Operations Guide, SC33-0510*
- *CICS/MVS Problem Determination Guide, SC33-0516*
- *CICS/MVS Resource Definition (Macro), SC33-0509*
- *CICS/MVS Resource Definition (Online), SC33-0508*

IBM C/C++ for MVS/ESA or OS/390

- *IBM C/C++ for MVS/ESA Library Reference, SC09-1995*
- *IBM C/C++ for MVS/ESA Programming Guide, SC09-1994*
- *IBM C/C++ for OS/390 User's Guide, SC09-2361*

IBM COBOL for MVS & VM

- *IBM COBOL for MVS & VM Language Reference, SC26-4769*
- *IBM COBOL for MVS & VM Programming Guide, SC26-4767*

Conversion Guides

- *DBMS Conversion Guide: DATACOM/DB to DB2, GH20-7564*
- *DBMS Conversion Guide: IDMS to DB2, GH20-7562*
- *DBMS Conversion Guide: Model 204 to DB2 or SQL/DS, GH20-7565*
- *DBMS Conversion Guide: VSAM to DB2, GH20-7566*
- *IMS-DB and DB2 Migration and Coexistence Guide, GH21-1083*

Cooperative Development Environment

- *CoOperative Development Environment/370: Debug Tool, SC09-1623*

DATABASE 2 for Common Servers

- *DATABASE 2 Administration Guide for common servers, S20H-4580*
- *DATABASE 2 Application Programming Guide for common servers, S20H-4643*
- *DATABASE 2 Software Developer's Kit for AIX: Building Your Applications, S20H-4780*
- *DATABASE 2 Software Developer's Kit for OS/2: Building Your Applications, S20H-4787*
- *DATABASE 2 SQL Reference for common servers, S20H-4665*
- *DATABASE 2 Call Level Interface Guide and Reference for common servers, S20H-4644*

Data Extract (DXT)

- *Data Extract Version 2: General Information, GC26-4666*
- *Data Extract Version 2: Planning and Administration Guide, SC26-4631*

DataPropagator NonRelational

- *DataPropagator NonRelational MVS/ESA Administration Guide, SH19-5036*
- *DataPropagator NonRelational MVS/ESA Reference, SH19-5039*

DataPropagator Relational

- *DataPropagator Relational User's Guide, SC26-3399*
- *IBM An Introduction to DataPropagator Relational, GC26-3398*

Data Facility Data Set Services

- *Data Facility Data Set Services: User's Guide and Reference, SC26-4125*

Database Design

- *DB2 Database Design and Implementation Using DB2, SH24-6101*
- *DB2 Design and Development Guide, Gabrielle Wiorkowski and David Kull, Addison Wesley*
- *Handbook of Relational Database Design, C. Fleming and B Von Halle, Addison Wesley*
- *Principles of Database Systems, Jeffrey D. Ullman, Computer Science Press*

DataHub

- *IBM DataHub General Information, GC26-4874*

DB2 Universal Database

- *DB2 Universal Database Administration Guide, S10J-8157*
- *DB2 Universal Database API Reference, S10J-8167*
- *DB2 Universal Database Application Development Guide, SC09-2845*
- *DB2 Universal Database Building Applications for UNIX Environments, S10J-8161*
- *DB2 Universal Database Building Applications for Windows and OS/2 Environments, S10J-8160*
- *DB2 Universal Database CLI Guide and Reference, S10J-8159*
- *DB2 Universal Database SQL Reference, S10J-8165*

Device Support Facilities

- *Device Support Facilities User's Guide and Reference, GC35-0033*

DFSMS/MVS

- *DFSMS/MVS: Access Method Services for the Integrated Catalog, SC26-4906*
- *DFSMS/MVS: Access Method Services for VSAM Catalogs, SC26-4905*
- *DFSMS/MVS: Administration Reference for DFSMSdss, SC26-4929*
- *DFSMS/MVS: DFSMShsm Managing Your Own Data, SH21-1077*
- *DFSMS/MVS: Diagnosis Reference for DFSMSdfp, LY27-9606*
- *DFSMS/MVS: Macro Instructions for Data Sets, SC26-4913*
- *DFSMS/MVS: Managing Catalogs, SC26-4914*
- *DFSMS/MVS: Program Management, SC26-4916*
- *DFSMS/MVS: Storage Administration Reference for DFSMSdfp, SC26-4920*
- *DFSMS/MVS: Using Advanced Services for Data Sets, SC26-4921*
- *DFSMS/MVS: Utilities, SC26-4926*
- *MVS/DFP: Managing Non-VSAM Data Sets, SC26-4557*

DFSORT

- *DFSORT Application Programming: Guide, SC33-4035*

Distributed Relational Database

- *Data Stream and OPA Reference, SC31-6806*
- *Distributed Relational Database Architecture: Application Programming Guide, SC26-4773*
- *Distributed Relational Database Architecture: Connectivity Guide, SC26-4783*
- *Distributed Relational Database Architecture: Evaluation and Planning Guide, SC26-4650*
- *Distributed Relational Database Architecture: Problem Determination Guide, SC26-4782*
- *Distributed Relational Database: Every Manager's Guide, GC26-3195*
- *IBM SQL Reference, SC26-8416*
- *Open Group Technical Standard (the Open Group presently makes the following books available through their website at www.opengroup.org):*
  - *DRDA Volume 1: Distributed Relational Database Architecture (DRDA), ISBN 1-85912-295-7*
  - *DRDA Volume 3: Distributed Database Management (DDM) Architecture, ISBN 1-85912-206-X*

Education

- *Dictionary of Computing, SC20-1699*
- *IBM Enterprise Systems Training Solutions Catalog, GR28-5467*

Enterprise System/9000 and Enterprise System/3090

- *Enterprise System/9000 and Enterprise System/3090 Processor Resource/System Manager Planning Guide, GA22-7123*

FORTRAN

- *VS FORTRAN Version 2: Language and Library Reference, SC26-4221*
- *VS FORTRAN Version 2: Programming Guide for CMS and MVS, SC26-4222*

High Level Assembler

- *High Level Assembler/MVS and VM and VSE Language Reference, SC26-4940*
- *High Level Assembler/MVS and VM and VSE Programmer's Guide, SC26-4941*

Parallel Sysplex Library

- *System/390 MVS Sysplex Application Migration, GC28-1211*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*
- *System/390 MVS Sysplex Overview: An Introduction to Data Sharing and Parallelism, GC28-1208*
- *System/390 MVS Sysplex Systems Management, GC28-1209*
- *System/390 MVS 9672/9674 System Overview, GA22-7148*

ICSF/MVS

- *ICSF/MVS General Information, GC23-0093*

IMS/ESA

- *IMS Batch Terminal Simulator General Information, GH20-5522*
- *IMS/ESA Administration Guide: System, SC26-8013*
- *IMS/ESA Application Programming: Database Manager, SC26-8727*
- *IMS/ESA Application Programming: Design Guide, SC26-8016*
- *IMS/ESA Application Programming: Transaction Manager, SC26-8729*
- *IMS/ESA Customization Guide, SC26-8020*
- *IMS/ESA Installation Volume 1: Installation and Verification, SC26-8023*
- *IMS/ESA Installation Volume 2: System Definition and Tailoring, SC26-8024*
- *IMS/ESA Messages and Codes, SC26-8028*
- *IMS/ESA Operator's Reference, SC26-8030*
- *IMS/ESA Utilities Reference: System, SC26-8035*

ISPF

- *ISPF Version 4 Messages and Codes, SC34-4450*
- *ISPF Version 4 for MVS Dialog Management Guide, SC34-4213*
- *ISPF/PDF Version 4 for MVS Guide and Reference, SC34-4258*

- *ISPF and ISPF/PDF Version 4 for MVS Planning and Customization, SC34-4134*

Language Environment for MVS & VM

- *Language Environment for MVS & VM Concepts Guide, GC26-4786*
- *Language Environment for MVS & VM Debugging and Run-Time Messages Guide, SC26-4829*
- *Language Environment for MVS & VM Installation and Customization, SC26-4817*
- *Language Environment for MVS & VM Programming Guide, SC26-4818*
- *Language Environment for MVS & VM Programming Reference, SC26-3312*

MVS/ESA

- *MVS/ESA Analyzing Resource Measurement Facility Monitor I and Monitor II Reference and User's Guide, LY28-1007*
- *MVS/ESA Analyzing Resource Measurement Facility Monitor III Reference and User's Guide, LY28-1008*
- *MVS/ESA Application Development Reference: Assembler Callable Services for OpenEdition MVS, SC23-3020*
- *MVS/ESA Data Administration: Utilities, SC26-4516*
- *MVS/ESA Diagnosis: Procedures, LY28-1844*
- *MVS/ESA Diagnosis: Tools and Service Aids, LY28-1845*
- *MVS/ESA Initialization and Tuning Guide, SC28-1451*
- *MVS/ESA Initialization and Tuning Reference, SC28-1452*
- *MVS/ESA Installation Exits, SC28-1459*
- *MVS/ESA JCL Reference, GC28-1479*
- *MVS/ESA JCL User's Guide, GC28-1473*
- *MVS/ESA JES2 Initialization and Tuning Guide, SC28-1453*
- *MVS/ESA MVS Configuration Program, GC28-1615*
- *MVS/ESA Planning: Global Resource Serialization, GC28-1450*
- *MVS/ESA Planning: Operations, GC28-1441*
- *MVS/ESA Planning: Workload Management, GC28-1493*
- *MVS/ESA Programming: Assembler Services Guide, GC28-1466*
- *MVS/ESA Programming: Assembler Services Reference, GC28-1474*
- *MVS/ESA Programming: Authorized Assembler Services Guide, GC28-1467*
- *MVS/ESA Programming: Authorized Assembler Services Reference, Volumes 1-4, GC28-1475, GC28-1476, GC28-1477, GC28-1478*
- *MVS/ESA Programming: Extended Addressability Guide, GC28-1468*
- *MVS/ESA Programming: Sysplex Services Guide, GC28-1495*
- *MVS/ESA Programming: Sysplex Services Reference, GC28-1496*
- *MVS/ESA Programming: Workload Management Services, GC28-1494*
- *MVS/ESA Routing and Descriptor Codes, GC28-1487*
- *MVS/ESA Setting Up a Sysplex, GC28-1449*
- *MVS/ESA SPL: Application Development Guide, GC28-1852*
- *MVS/ESA System Codes, GC28-1486*
- *MVS/ESA System Commands, GC28-1442*
- *MVS/ESA System Management Facilities (SMF), GC28-1457*
- *MVS/ESA System Messages Volume 1, GC28-1480*
- *MVS/ESA System Messages Volume 2, GC28-1481*
- *MVS/ESA System Messages Volume 3, GC28-1482*
- *MVS/ESA Using the Subsystem Interface, SC28-1502*

Net.Data for OS/390

- *Net.Data Language Environment Guide, http://www.ibm.com/software/net.data/docs*
- *Net.Data Programming Guide, http://www.ibm.com/software/net.data/docs*
- *Net.Data Reference Guide, http://www.ibm.com/software/net.data/docs*

NetView

- *NetView Installation and Administration Guide, SC31-8043*
- *NetView User's Guide, SC31-8056*

ODBC

- *ODBC 2.0 Programmer's Reference and SDK Guide, ISBN 1-55615-658-8*
- *Inside ODBC, ISBN 1-55615-815-7*

OS/390

- *OS/390 C/C++ Programming Guide, SC09-2362*
- *OS/390 C/C++ Run-Time Library Reference, SC28-1663*
- *OS/390 Information Roadmap, GC28-1727*
- *OS/390 Introduction and Release Guide, GC28-1725*
- *OS/390 JES2 Initialization and Tuning Guide, SC28-1791*
- *OS/390 JES3 Initialization and Tuning Guide, SC28-1802*
- *OS/390 Language Environment for OS/390 & VM Concepts Guide, GC28-1945*
- *OS/390 Language Environment for OS/390 & VM Customization, SC28-1941*
- *OS/390 Language Environment for OS/390 & VM Debugging Guide, SC28-1942*
- *OS/390 Language Environment for OS/390 & VM Programming Guide, SC28-1939*
- *OS/390 Language Environment for OS/390 & VM Programming Reference, SC28-1940*
- *OS/390 MVS Diagnosis: Procedures, LY28-1082*
- *OS/390 MVS Diagnosis: 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 Security Server (RACF) Auditor's Guide, SC28-1916*
- *OS/390 Security Server (RACF) Command Language Reference, SC28-1919*
- *OS/390 Security Server (RACF) General User's Guide, SC28-1917*
- *OS/390 Security Server (RACF) Security Administrator's Guide, SC28-1915*
- *OS/390 Security Server (RACF) System Programmer's Guide, SC28-1913*
- *OS/390 SMP/E Reference, SC28-1806*
- *OS/390 SMP/E User's Guide, SC28-1740*
- *OS/390 RMF User's Guide, SC28-1949*
- *OS/390 TSO/E CLISTS, SC28-1973*
- *OS/390 TSO/E Command Reference, SC28-1969*
- *OS/390 TSO/E Customization, SC28-1965*
- *OS/390 TSO/E Messages, GC28-1978*
- *OS/390 TSO/E Programming Guide, SC28-1970*
- *OS/390 TSO/E Programming Services, SC28-1971*
- *OS/390 TSO/E REXX Reference, SC28-1975*
- *OS/390 TSO/E User's Guide, SC28-1968*

OS/390 OpenEdition

- *OS/390 OpenEdition DCE Administration Guide, SC28-1584*
- *OS/390 OpenEdition DCE Introduction, GC28-1581*
- *OS/390 R1 OE DCE Messages and Codes, ST01-0920*
- *OS/390 OpenEdition Command Reference, SC28-1892*
- *OS/390 OpenEdition Messages and Codes, SC28-1908*
- *OS/390 OpenEdition Planning, SC28-1890*
- *OS/390 OpenEdition User's Guide, SC28-1891*

PL/I for MVS & VM

- *IBM PL/I MVS & VM Language Reference, SC26-3114*
- *IBM PL/I MVS & VM Programming Guide, SC26-3113*

OS PL/I

- *OS PL/I Programming Language Reference, SC26-4308*
- *OS PL/I Programming Guide, SC26-4307*

PROLOG

- *IBM SAA AD/Cycle Prolog/MVS & VM Programmer's Guide, SH19-6892*

Query Management Facility

- *Query Management Facility: Managing QMF for MVS, SC26-8218*
- *Query Management Facility: Reference, SC26-4716*
- *Query Management Facility: Using QMF, SC26-8078*

Remote Recovery Data Facility

- *Remote Recovery Data Facility Program Description and Operations, LY37-3710*

Resource Access Control Facility (RACF)

- *External Security Interface (RACROUTE) Macro Reference for MVS and VM, GC28-1366*
- *Resource Access Control Facility (RACF) Auditor's Guide, SC28-1342*
- *Resource Access Control Facility (RACF) Command Language Reference, SC28-0733*

- *Resource Access Control Facility (RACF) General Information Manual, GC28-0722*
- *Resource Access Control Facility (RACF) General User's Guide, SC28-1341*
- *Resource Access Control Facility (RACF) Security Administrator's Guide, SC28-1340*
- *Recource Access Control Facility (RACF) System Programmer's Guide, SC28-1343*

Storage Management

- *MVS/ESA Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *MVS/ESA Storage Management Library: Leading an Effective Storage Administration Group, SC26-3126*
- *MVS/ESA Storage Management Library: Managing Data, SC26-3124*
- *MVS/ESA Storage Management Library: Managing Storage Groups, SC26-3125*
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide, SC26-4659*

System/370 and System/390

- *IBM System/370 ESA Principles of Operation, SA22-7200*
- *IBM System/390 ESA Principles of Operation, SA22-7205*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*

System Modification Program Extended (SMP/E)

- *System Modification Program Extended (SMP/E) Reference, SC28-1107*
- *System Modification Program Extended (SMP/E) User's Guide, SC28-1302*

System Network Architecture (SNA)

- *SNA Formats, GA27-3136*
- *SNA LU 6.2 Peer Protocols Reference, SC31-6808*
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2, GC30-3084*
- *SNA/Management Services Alert Implementation Guide, GC31-6809*

TCP/IP

- *IBM TCP/IP for MVS: Customization & Administration Guide, SC31-7134*
- *IBM TCP/IP for MVS: Diagnosis Guide, LY43-0105*
- *IBM TCP/IP for MVS: Messages and Codes, SC31-7132*
- *IBM TCP/IP for MVS: Planning and Migration Guide, SC31-7189*

TSO Extensions

- *TSO/E CLISTS, SC28-1876*
- *TSO/E Command Reference, SC28-1881*
- *TSO/E Customization, SC28-1872*
- *TSO/E Messages, GC28-1885*
- *TSO/E Programming Guide, SC28-1874*
- *TSO/E Programming Services, SC28-1875*
- *TSO/E User's Guide, SC28-1880*

VS COBOL II

- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, SC26-4047*
- *VS COBOL II Installation and Customization for MVS, SC26-4048*

VTAM

- *Planning for NetView, NCP, and VTAM, SC31-8063*
- *VTAM for MVS/ESA Diagnosis, LY43-0069*
- *VTAM for MVS/ESA Messages and Codes, SC31-6546*
- *VTAM for MVS/ESA Network Implementation Guide, SC31-6548*
- *VTAM for MVS/ESA Operation, SC31-6549*
- *VTAM for MVS/ESA Programming, SC31-6550*
- *VTAM for MVS/ESA Programming for LU 6.2, SC31-6551*
- *VTAM for MVS/ESA Resource Definition Reference, SC31-6552*

Index

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

We'd Like to Hear from You

DB2 for OS/390
Version 5
Administration Guide

Publication No. SC26-8957-03

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773 or (408) 463-4393.
- Electronic mail—Use one of the following network IDs:
  - IBMMail: USIBMXFC @ IBMMAIL
  - IBMLink: DB2PUBS @ STLVM27
  - Internet: DB2PUBS@VNET.IBM.COM

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number or your name and electronic address if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

DB2 for OS/390
Version 5
Administration Guide

Publication No. SC26-8957-03

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Address

Company or Organization

Phone No.

Readers' Comments
SC26-8957-03

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department BWE/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape | **Please do not staple** | Fold and Tape

SC26-8957-03

Cut or Fold
Along Line

Program Number: 5655-DB2

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

DB2 for OS/390
Version 5

SC26-8957	Administration Guide
SC26-8958	Application Programming and SQL Guide
SC26-8959	Call Level Interface Guide and Reference
SC26-8960	Command Reference
SC26-8961	Data Sharing: Planning and Administration
SX26-3841	Data Sharing Quick Reference
LY27-9659	Diagnosis Guide and Reference
LY27-9660	Diagnosis Quick Reference
GC26-8970	Installation Guide
GC26-8979	Master Index
SC26-8962	Messages and Codes
SC26-8964	Reference for Remote DRDA Requesters and Servers
SX26-3842	Reference Summary
SC26-8965	Release Guide
SC26-8966	SQL Reference
SC26-8967	Utility Guide and Reference
GC26-8971	What's New?

SC26-8957-03