IBM DB2 Universal Database Administration Guide Version 5

Document Number S10J-8157-00

Authors:

IBM Toronto Lab

IBM DB2 Universal Database

Administration Guide

Version 5

S10J-8157-00

IBM DB2 Universal Database

Administration Guide

Version 5

S10J-8157-00

Before using this information and the product it supports, be sure to read the general information under Appendix R, “Notices” on page 873.

Order publications through your IBM representative or the IBM branch office serving your locality or by calling 1-800-879-2755 in U.S. or 1-800-IBM-4YOU in Canada.

Contents

About This Book

This book provides information necessary to use and administer the DB2* relational database management system (RDBMS) products, including:

- Information required for designing, implementing and managing databases
- Information regarding the configuring and tuning of your database environment to improve performance.

Many of the tasks described in this book can be performed using different interfaces:

- The **Command Processor**, which allows you to access and manipulate databases from a graphical interface. From this interface, you can also execute SQL statements and DB2 utility functions. Most examples in this book illustrate the use of this interface. For more information about using the command processor, see the *Command Reference* manual.
- The **application programming interface**, which allows you to execute DB2 utility functions within an application program. For more information about using the application programming interface, see the *API Reference* manual.
- The **Control Center**, which allows you to graphically perform administrative tasks such as configuring the system, managing directories, backing up and recovering the system, scheduling jobs, and managing media. The Control Center also contains `Replication Administration` to graphically setup the replication of data between systems. execute DB2 utility functions through a graphical user interface. To invoke the Control Center, use the `db2cc` command, or (for OS/2) select the Control Center icon from the DB2 folder. For introductory help, select **Getting started** from the **Help** pull-down of the Control Center window. The **Visual Explain** and **Performance Monitor** tools are invoked from the Control Center.

There are other tools available that you can use to perform administration tasks. They include:

- The `Script Center` to store small applications called scripts. These scripts may contain DB2 commands as well as operating system commands.
- The `Alert Center` to monitor the messages that result from other DB2 operations.
- The `Tool Settings` to change the settings for the Control Center, Alert Center, and Replication.
- The `Journal` to schedule jobs to run unattended.

Who Should Use this book

This book is intended primarily for database administrators, system administrators, security administrators and system operators who need to design, implement and maintain a database to be accessed by local or remote clients. It can also be used by programmers and other users who require an understanding of the administration and operation of the DB2 relational database management system.

How This Book is Structured

The *Administration Guide* contains information about the following major topics:

- Introduction to Parallelism in DB2 Universal Database, presents an overview of DB2 Universal Database and the types of parallelism provided by DB2.

Database Design and Implementation

- Chapter 1, Designing Your Logical Database, discusses the concepts and guidelines for designing a logical database.
- Chapter 2, Designing Your Physical Database, discusses the guidelines for designing a physical database, including considerations related to physical data storage.
- Chapter 3, Implementing Your Design, discusses the concepts and guidelines for creating a database and the objects within a database.
- Chapter 4, Controlling Database Access, describes how you can control access to your database's resources.
- Chapter 5, Utilities for Moving Data, discusses the LOAD, AutoLoader, IMPORT and EXPORT utilities. db2move and replication are also discussed.
- Chapter 6, Recovering a Database, discusses factors to consider when choosing database and table space recovery methods, including backing up and restoring a database or table space, and using the roll-forward recovery method.

Distributed Transaction Processing

- Chapter 7, Distributed Databases, discusses how you can access multiple databases in a single transaction.
- Chapter 8, Using DB2 with an XA-Compliant Transaction Manager, discusses how you can use your databases in a distributed transaction processing environment such as CICS.

Tuning Application Performance

- Chapter 9, Application Considerations, describes some techniques for improving database performance when designing your applications.
- Chapter 10, Environmental Considerations, describes some techniques for improving database performance when setting up your database environment.
- Chapter 11, System Catalog Statistics, describes how statistics about your data can be collected and used to ensure optimal performance.
- Chapter 12, Understanding the SQL Compiler, describes what happens to an SQL statement when it is compiled using the SQL compiler.
- Chapter 13, SQL Explain Facility, describes the Explain facility, which allows you to examine the choices the SQL compiler has made to access your data.

Tuning and Configuring Your System

- Chapter 14, Operational Performance, provides an overview of how the database manager uses memory and other considerations that affect run-time performance.
- Chapter 15, Using the Governor, provides an introduction to the use of a governor to control some aspects of database management.
- Chapter 16, Redistributing Data Across Database Partitions, discusses the tasks required in a partitioned database environment to redistribute data across partitions.
- Chapter 17, Scaling Your Configuration, introduces some considerations and tasks associated with increasing the size of your database systems.
- Chapter 18, Benchmark Testing, provides an overview of benchmark testing and how to perform benchmark testing.
- Chapter 19, Configuring DB2, discusses the database manager and database configuration files and the values for the configuration parameters.

Appendixes

- Appendix A, Planning Database Migration, provides information about migrating databases to Version 5.
- Appendix B, Incompatibilities Between Releases, presents the incompatibilities introduced with Version 5.
- Appendix C, Memory Usage for DB2 Universal Database Version 5, presents memory requirements for each DB2 feature.
- Appendix D, Naming Rules, provides the rules to follow when naming databases and objects.
- Appendix E, DB2 Registry Values and Environment Variables, presents profile registry values and environment variables.
- Appendix F, Using Distributed Computing Environment (DCE) Directory Services, provides information about how you can use DCE Directory Services.
- Appendix G, X/Open Distributed Transaction Processing Model, provides an overview of the X/Open Distributed Transaction Processing model and the DB2 database support provided.
- Appendix H, Sample Tables, contains a description of the sample tables provided with the database manager.
- Appendix I, Catalog Views, contains a description of each system catalog view, including column names and data types.
- Appendix J, User Exit for Database Recovery, discusses how user exit programs can be used with database log files and describes some sample user exit programs.
- Appendix K, Explain Tables and Definitions, provides information about the tables used by the DB2 Explain facility and how to create those tables.
- Appendix L, SQL Explain Tools (db2expln and dynexpln), provides information on using the DB2 explain tools: db2expln and dynexpln.

- Appendix M, National Language Support (NLS), introduces DB2 National Language Support (NLS) including information about countries, languages, and code pages.
- Appendix N, Splitting Data with db2split, discusses how to use the db2split tool in a partitioned database environment.
- Appendix O, Supplemental AutoLoader Information, presents supplmental AutoLoader information. AutoLoader is only for use in a partitioned database environment.
- Appendix P, Issuing Commands to Multiple Database Partitions, discusses the use of the *db2_all* and *rah* shell scripts to send commands to all partitions in a partitioned database environment.
- Appendix Q, Supporting High Availability Cluster Multi-Processing Configurations, discusses the support of IBM High Availability Cluster Multi-Processing (HACMP) for AIX by DB2.

Introduction to Parallelism in DB2 Universal Database

This chapter provides an introduction to DB2 Universal Database and to the types of parallelism provided by DB2. This chapter describes the following:

- Overview of basic DB2 concepts and DB2 parallelism concepts
- Types of parallelism
- Hardware environments
- Summary of parallelism possible for each hardware environment
- Enabling parallelism

DB2 provides the flexibility for you to run a wide range of hardware configurations. It allows you to choose how to best match your hardware and application requirements with a specific DB2 product configuration.

The remaining chapters in this part of the book assist you in the design and implementation of your database. With the different levels of complexity in database environments that DB2 supports, there are considerations and tasks specific to one or more of these environments. These considerations and tasks are presented toward the end of each section or chapter and introduced as being for a specific environment. In some cases, entire sections or chapters are appropriate for only a specific environment. After reading this chapter, you should be able to discern which chapters are appropriate for your business needs and your environment.

Overview of DB2 Concepts

A *database manager* (sometimes called an *instance*) is DB2 code that manages data. It controls what can be done to the data, and manages system resources assigned to it. Each instance is a complete environment. It contains all the database partitions defined for a given parallel database system. An instance has its own databases (which other instances cannot access), and all its database partitions share the same system directories. It also has separate security from other instances on the same machine.

A *table* consists of data logically arranged in columns and rows. The data in the table is logically related, and relationships can be defined between tables. Data can be viewed and manipulated based on mathematical principles and operations called relations. Table data is accessed via SQL, a standardized language for defining and manipulating data in a relational database. All database and table data is assigned to table spaces.

A database is organized into parts called *table spaces*. A table space's definition and attributes are recorded in the database system catalog. Once a table space is created, you can then create tables within this table space. A container is assigned to a table space. A *container* is an allocation of physical storage (such as a file or device). Table spaces reside in nodegroups.

A *nodegroup* is a set of one or more database partitions. When you want to create tables for the database, you first create the nodegroup where the table spaces will be

stored, then you create the table space where the tables will be stored. See “Nodegroups and Data Partitioning” on page xxvii for more information about nodegroups. See “Overview of DB2 Parallelism Concepts” on page xxvii for the definition of a database partition.

Figure 1 illustrates the relationship among the objects just described. It also illustrates that tables, indexes, and long data are stored in table spaces.

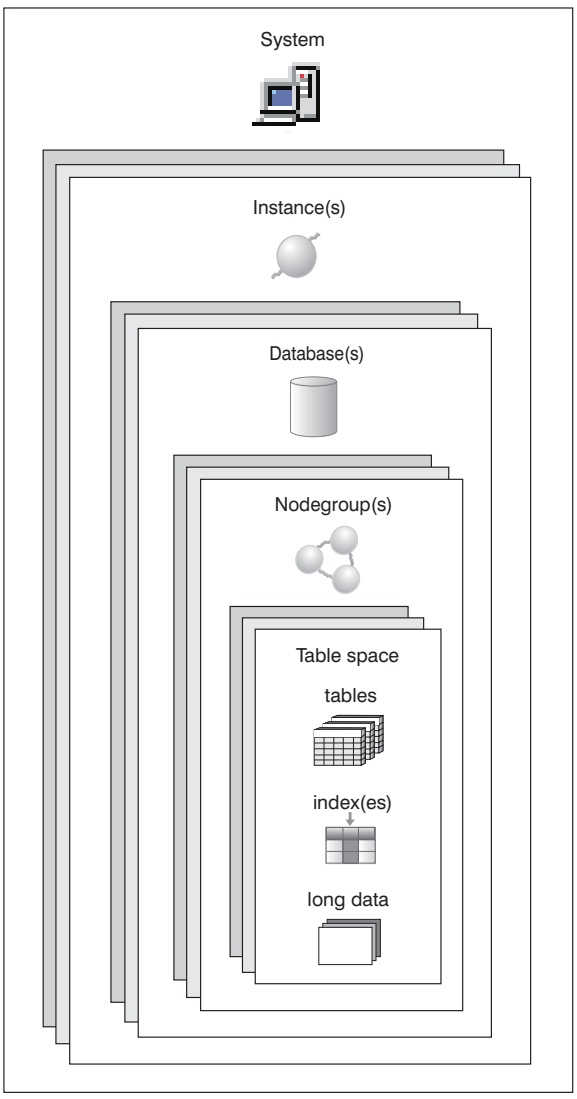


Figure 1. Relationship Among Some Database Objects

Overview of DB2 Parallelism Concepts

DB2 extends the database manager to the parallel, multi-node environment. A *database partition* is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. (See the *Administration Getting Started* for an overview of indexes, configuration files, and transaction logs.) A database partition is sometimes called a node or database node. (Node was the term used in the DB2 Parallel Edition for AIX Version 1 product.)

A *single-partition database* is a database having only one database partition. All data in the database is stored in that partition. In this case nodegroups, while present, provide no additional capability.

A *partitioned database* is a database with two or more database partitions. Tables can be located in one or more database partitions. When a table is in a nodegroup consisting of multiple partitions, some of its rows are stored in one partition and others are stored in other partitions.

Usually, a single database partition exists on each physical node and the processors on each system are used by the database manager at each database partition to manage its part of the database's total data.

Because data is divided across database partitions, you can use the power of multiple processors on multiple physical nodes to satisfy requests for information. Data retrieval and update requests are decomposed automatically into sub-requests and executed in parallel among the applicable database partitions. The fact that databases are split across database partitions is transparent to users of SQL statements.

User interaction is through one database partition. It is known as the *coordinator node* for that user. The coordinator runs on the same database partition as the application, or in the case of a remote application, the database partition to which that application is connected. Any database partition can be used as a coordinator node.

Nodegroups and Data Partitioning

You can define named subsets of one or more database partitions in a database. Each subset you define is known as a *nodegroup*. Each subset that contains more than one database partition is known as a *multi-partition nodegroup*. Multi-partition nodegroups can only be defined with database partitions that belong to the same instance.

Figure 2 on page xxviii shows an example of a database with five partitions in which:

- A nodegroup spans all but one of the database partitions (Nodegroup 1)
- A nodegroup contains one database partition (Nodegroup 2)
- A nodegroup contains one database partition which acts as the coordinator node (Nodegroup 3)

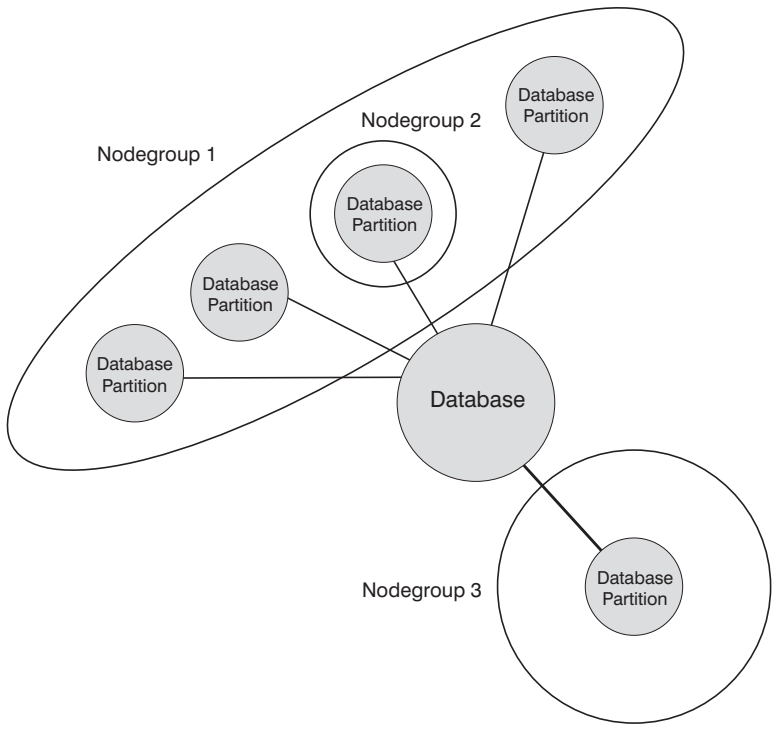


Figure 2. Nodegroups in a Database

You create a new nodegroup using the CREATE NODEGROUP statement. See the *SQL Reference* for more information. Data is divided across all the partitions in a nodegroup. If you are using a multi-partition nodegroup, you must look at several nodegroup design considerations. For more information in both of these areas, see "Designing Nodegroups" on page 32.

Types of Parallelism

Parts of a database-related task (such as a database query) can be executed in parallel in order to speed up the task, often dramatically so. There are different ways a task is performed in parallel. The nature of the task, the database configuration, and the hardware environment determine how DB2 will perform a task in parallel. These considerations are interrelated. You should consider them together when first deciding on the physical and logical design of a database. This section describes the types of parallelism.

DB2 supports the following types of parallelism:

- I/O
- Query
- Utility

I/O Parallelism

For situations in which multiple containers exist for a table space, the database manager can initiate *parallel I/O*. Parallel I/O refers to the process of reading from or writing to two or more I/O devices at the same time to reduce elapsed time. Performing I/O in parallel can result in significant improvements to I/O throughput.

I/O parallelism is a component of each hardware environment described in “Hardware Environments” on page xxxiii. Table 1 on page xli lists the hardware environments best suited for I/O parallelism.

Query Parallelism

There are two types of query parallelism: inter-query parallelism and intra-query parallelism.

Inter-query parallelism refers to the ability of multiple applications to query a database at the same time. Each query will execute independently of the others, but DB2 will execute all of them at the same time. DB2 has always supported this type of parallelism.

Intra-query parallelism refers to the processing of parts of a single query at the same time using either *intra-partition parallelism* or *inter-partition parallelism* or both.

The term *query parallelism* is used throughout this book.

Intra-Partition Parallelism

Intra-partition parallelism refers to the ability to break up a query into multiple parts. (Some of the utilities also perform this type of parallelism. See “Utility Parallelism” on page xxxii.)

Intra-partition parallelism subdivides what is usually considered a single database operation such as index creation, database load, or SQL queries into multiple parts, many or all of which can be executed in parallel *within a single database partition*.

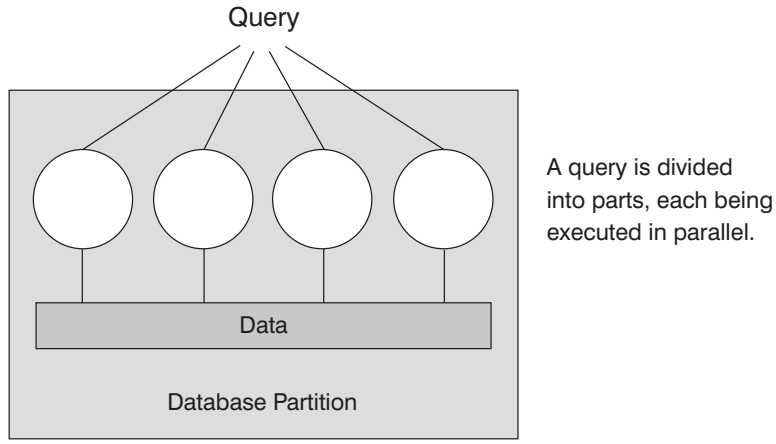


Figure 3. Intra-Partition Parallelism

Figure 3 shows a query that is broken into four pieces that can be executed in parallel, with the results returned more quickly than if the query was run in a serial fashion. The pieces are copies of each other. To utilize intra-partition parallelism, you need to configure the database appropriately.

You can choose the degree of parallelism or let the system do it for you. The degree of parallelism is the number of pieces of a query that execute in parallel.

Table 1 on page xli lists the hardware environments best suited for intra-partition parallelism.

Inter-Partition Parallelism

Inter-partition parallelism refers to the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one machine or multiple machines. The query is performed in parallel. (Some of the utilities also perform this type of parallelism. See “Utility Parallelism” on page xxxii.)

Inter-partition parallelism subdivides what is usually considered a single database operation such as index creation, database load, or SQL queries into multiple parts, many or all of which can be executed in parallel *across multiple partitions of a partitioned database* in one machine or multiple machines.

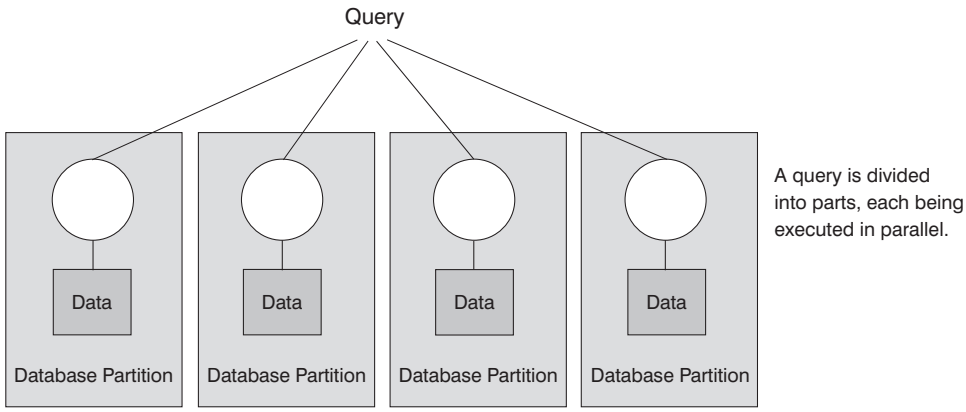


Figure 4. Inter-Partition Parallelism

Figure 4 shows a query that is broken into four pieces that can be executed in parallel, with the results returned more quickly than if the query was run in a serial fashion in a single partition.

The degree of parallelism is largely determined by the number of partitions you create and how you define your nodegroups.

Table 1 on page xli lists the hardware environments best suited for inter-partition parallelism.

Using Both Intra-Partition and Inter-Partition Parallelism

You can use intra-partition parallelism and inter-partition parallelism at the same time. This combination provides, in effect, two dimensions of parallelism. This results in an even more dramatic increase in the speed at which queries are processed. Figure 5 on page xxxii illustrates this.

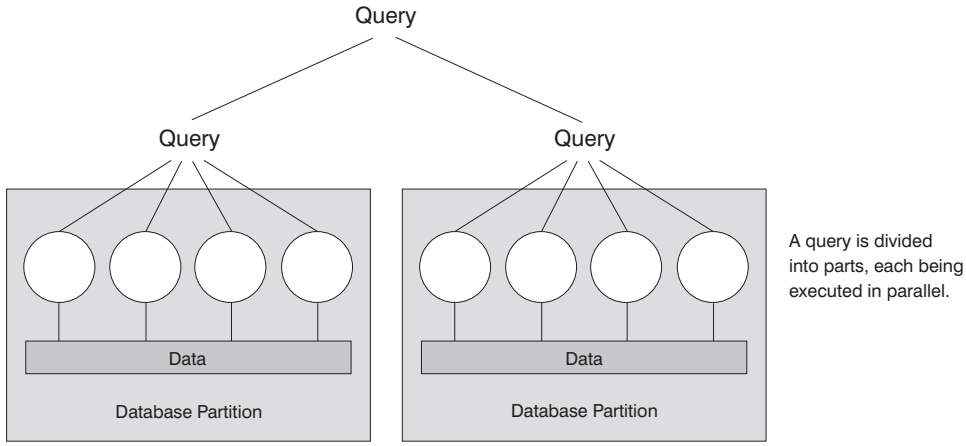


Figure 5. Both Inter-Partition and Intra-Partition Parallelism

Utility Parallelism

DB2's utilities can take advantage of intra-partition parallelism. They can also take advantage of inter-partition parallelism; where multiple database partitions exist, the utilities execute in each of the partitions in parallel. The following paragraphs describe how some utilities take advantage of parallelism.

The LOAD utility can take advantage of intra-partition parallelism and I/O parallelism. Loading data is a heavily CPU-intensive task. The LOAD utility takes advantage of multiple processors for tasks such as parsing and formatting data. Also, the LOAD utility can use parallel I/O servers to write the data to the containers in parallel. See "LOAD Performance Considerations" on page 148 and the LOAD command in the *Command Reference* for information on how to enable parallelism for the LOAD utility.

During index creation, the scanning and subsequent sorting of the data occurs in parallel. DB2 exploits both I/O parallelism and intra-partition parallelism when creating an index. This helps to speed up index creation when a CREATE INDEX command is issued, during restart (if an index is marked invalid), and during the reorganization of data.

Backing up and restoring data are heavily I/O bound tasks. DB2 exploits both I/O parallelism and intra-partition parallelism when performing backups and restores. Backup exploits I/O parallelism by reading from multiple table space containers in parallel, and asynchronously writing to multiple backup media in parallel. See the BACKUP DATABASE command and the RESTORE DATABASE command in the *Command Reference* for information on how to enable parallelism for these two commands.

Hardware Environments

This section provides an overview of the following hardware environments:

- Single partition on a single processor (uniprocessor)
- Single partition with multiple processors (SMP)
- Multiple partition configurations
  - Partitions with one processor (MPP)
  - Partitions with multiple processors (cluster of SMPs)
  - Logical database partitions (also known as Multiple Logical Nodes (MLN) in DB2 Parallel Edition for AIX Version 1)

In each hardware environment section, considerations for capacity and scalability are described. *Capacity* refers to the number of users and applications able to access the database in large part determined by memory, agents, locks, I/O, and storage management. *Scalability* refers to the ability for a database to grow and continue to exhibit the same operating characteristics and response times.

Single Partition on a Single Processor

This environment is made up of memory and disk, but contains only a single CPU. This environment has been given many names such as: standalone database, client/server database, serial database, uniprocessor system, and single node/non-parallel environment. Figure 6 on page xxxiv illustrates this environment.

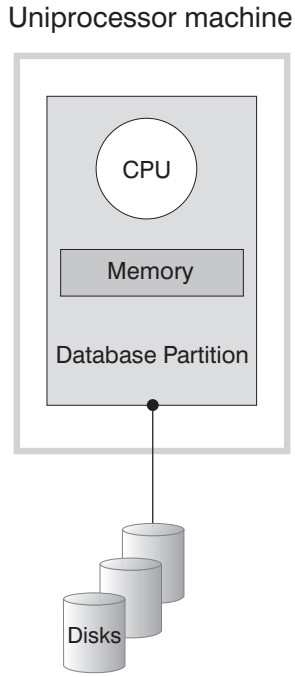


Figure 6. Single Partition On a Single Processor

The database in this environment serves the needs of a department or small office where the data and system resources (including only a single processor or CPU) are managed by a single database manager.

Table 1 on page xli lists the types of parallelism best suited to take advantage of this hardware configuration.

Capacity and Scalability

In this environment you can add more disks. Having one or more I/O servers for each disk allows for more than one I/O operation to be taking place at the same time. You can also add more hard disk space to this environment.

A single-processor system is restricted by the amount of disk space the processor can handle. However, as workload increases a single CPU may become insufficient in processing user requests any faster, regardless of other additional components, such as memory or disk, that you may add.

If you have reached maximum capacity or scalability, you can consider moving to a single partition system with multiple processors. This configuration is described in the next section.

Single Partition with Multiple Processors

This environment is typically made up of several equally powerful processors within the same machine and is called a *symmetric multi-processor (SMP)* system. Resources such as disk space and memory are *shared*. More disks and memory are found in this machine compared to the single-partition database, single processor environment. This environment is easy to manage since physically everything is together in one machine and the sharing of memory and disks is expected.

With multiple processors available, different database operations can be completed significantly more quickly than with databases assigned to only a single processor. DB2 can also divide the work of a single query among available processors to improve processing speed. Other database operations such as the LOAD utility, the backup and restore of table spaces, and index creation on existing data can take advantage of the multiple processors. Figure 7 illustrates this environment.

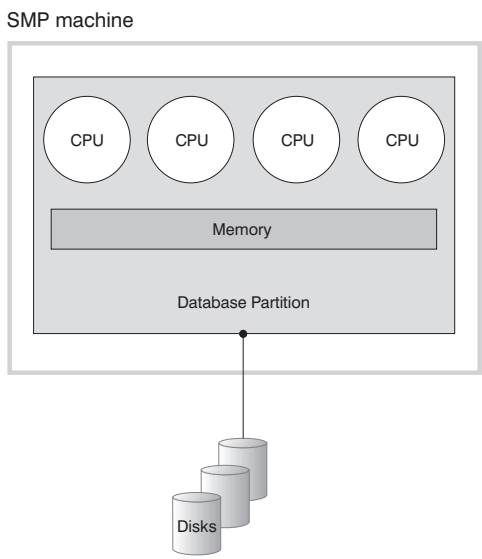


Figure 7. Single Partition Database Symmetric Multiprocessor System

Table 1 on page xli lists the types of parallelism best suited to take advantage of this hardware configuration.

Capacity and Scalability

In this environment you can add more processors. However, since it is possible for the different processors to attempt to access the same data, limitations with this environment can appear as your business operations grow. With shared memory and shared disks, you are effectively sharing all of the database data. One application on one processor may be accessing the same data as another application on another processor, possibly causing the second application to wait for access to the data.

You can increase the I/O capacity of the database partition associated with your processor, such as the number of disks. You can establish I/O servers to specifically deal with I/O requests. Having one or more I/O servers for each disk allows for more than one I/O operation to take place at the same time.

If you have reached maximum capacity or scalablity, you can consider moving to a system with multiple partitions. These configurations are described in the next section.

Multiple Partition Configurations

You can divide a database into multiple partitions, each on its own machine. Multiple machines with multiple database partitions can be grouped together. This section describes the following partition configurations:

- Partitions on systems each with one processor
- Partitions on systems each with multiple processors
- Logical database partitions

Partitions with One Processor

In this environment there are many database partitions with each partition on its own machine and having its own processor, memory, and disks. Figure 8 on page xxxvii illustrates this. A machine consists of a CPU, memory, and disk with all machines connected by a communications facility. Other names that have been given to this environment include: a cluster, a cluster of uniprocessors, a massively parallel processing (MPP) environment, or a shared-nothing configuration. The latter name accurately reflects the arrangement of resources in this environment. Unlike an SMP environment, an MPP environment has no shared memory or disks. The MPP environment removes the limitations introduced through the sharing of memory and disks.

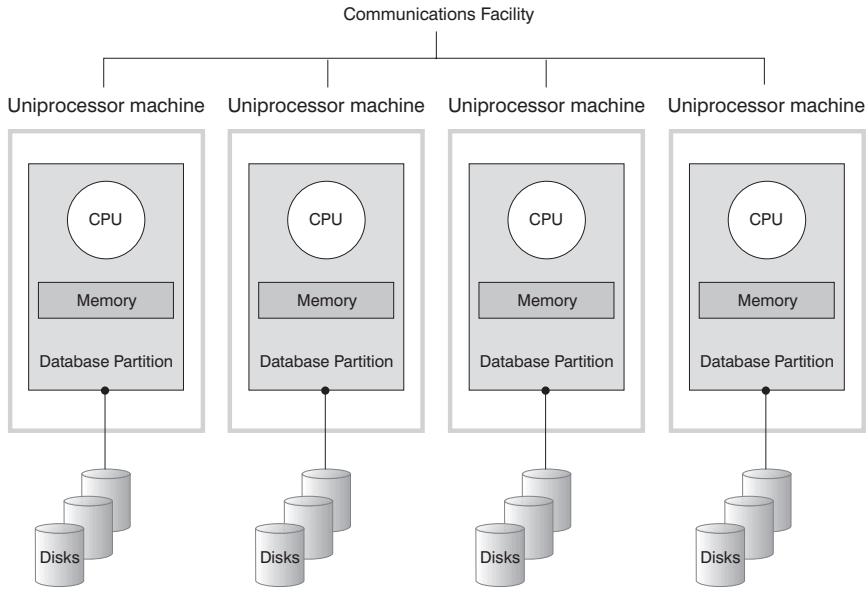


Figure 8. Massively Parallel Processing System

A partitioned database environment allows a database to remain a logical whole while being physically divided across more than one partition. To applications or users, the database can be used as a whole and the fact that data is partitioned remains transparent to most users. The work to be done with the data can be divided out to each of the database managers. Each database manager in each partition works against its own part of the database.

Table 1 on page xli lists the types of parallelism best suited to take advantage of this hardware configuration.

Capacity and Scalability: In this environment you can add more database partitions (nodes) to your configuration. On some platforms, for example the RS/6000 SP, the maximum is 512 nodes. However, there may be practical limits for managing a high number of machines and instances.

If you have reached maximum capacity or scalability, you can consider moving to a system where each partition has multiple processors. This configuration is described in the next section.

Partitions with Multiple Processors

As an alternative to a configuration in which each partition has a single processor is a configuration in which a partition has multiple processors. This is known as an *SMP cluster*.

This configuration combines the advantages of SMP and MPP parallelism. This means a query can be performed in a single partition across multiple processors. It also means that a query can be performed in parallel across multiple partitions.

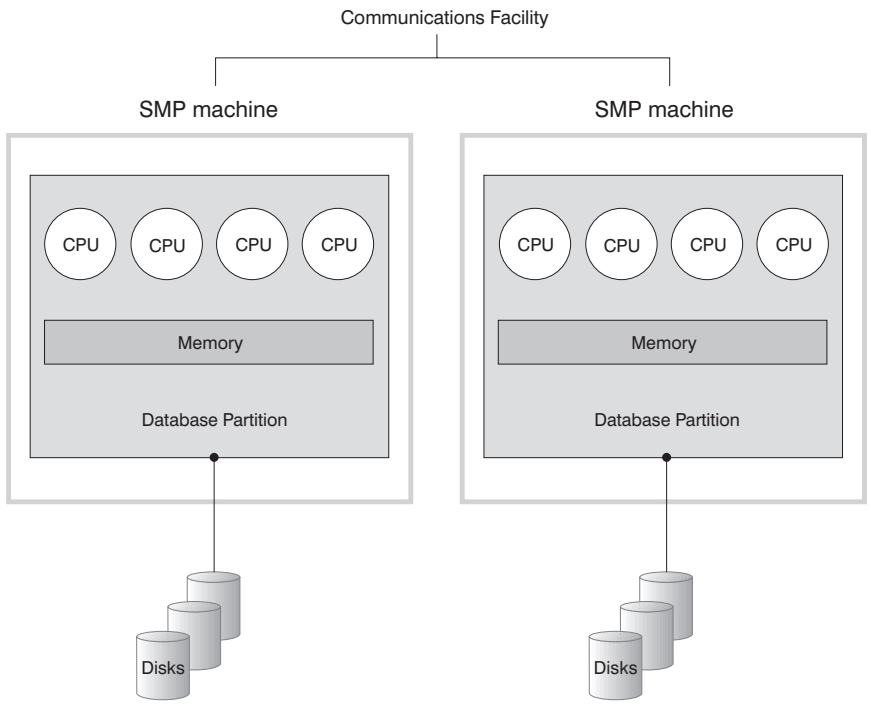


Figure 9. Cluster of SMPs

Table 1 on page xli lists the types of parallelism best suited to take advantage of this hardware configuration.

Capacity and Scalability: In this environment you can add more database partitions, as in the previous section. You can also add more processors to existing database partitions.

Logical Database Partitions

A logical database partition differs from a physical partition in that it is not given control of an entire machine. Although the machine has shared resources, the database partitions do not share the resources. Processors are shared but disk and memory are not.

One reason for using logical database partitions is to provide scalability. Multiple database managers running in multiple logical partitions may be able to make fuller use of available resources than a single database manager could. This will become more true as machines with even more more processors are manufactured. Figure 10 on

page xxxix illustrates the fact that you may gain more scalability on an SMP machine by adding more partitions, particularly for machines with many processors. By partitioning the database, you can administer and recover each partition separately.

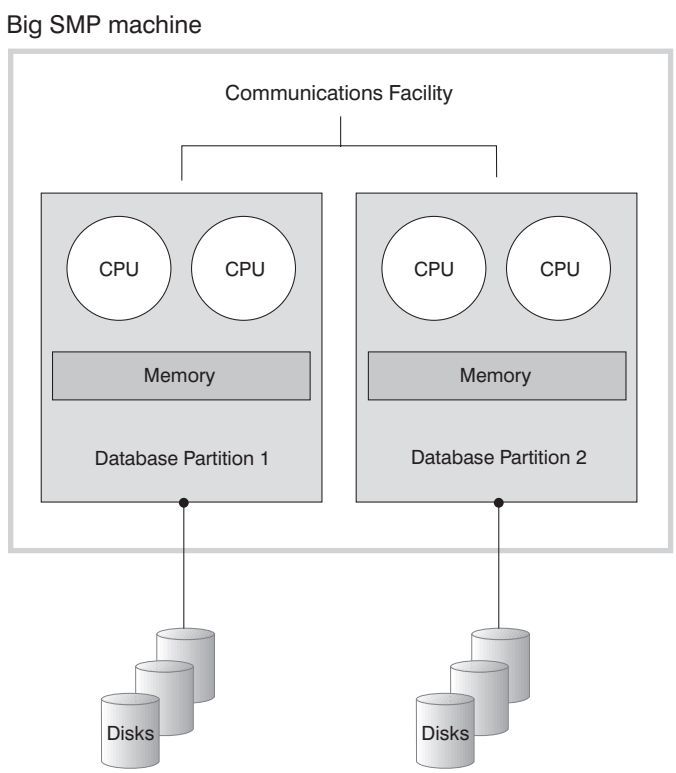


Figure 10. Partitioned Database, Symmetric Multiprocessor System

Figure 11 on page xl illustrates the fact that you can multiply the configuration shown in Figure 10 to increase processing power.

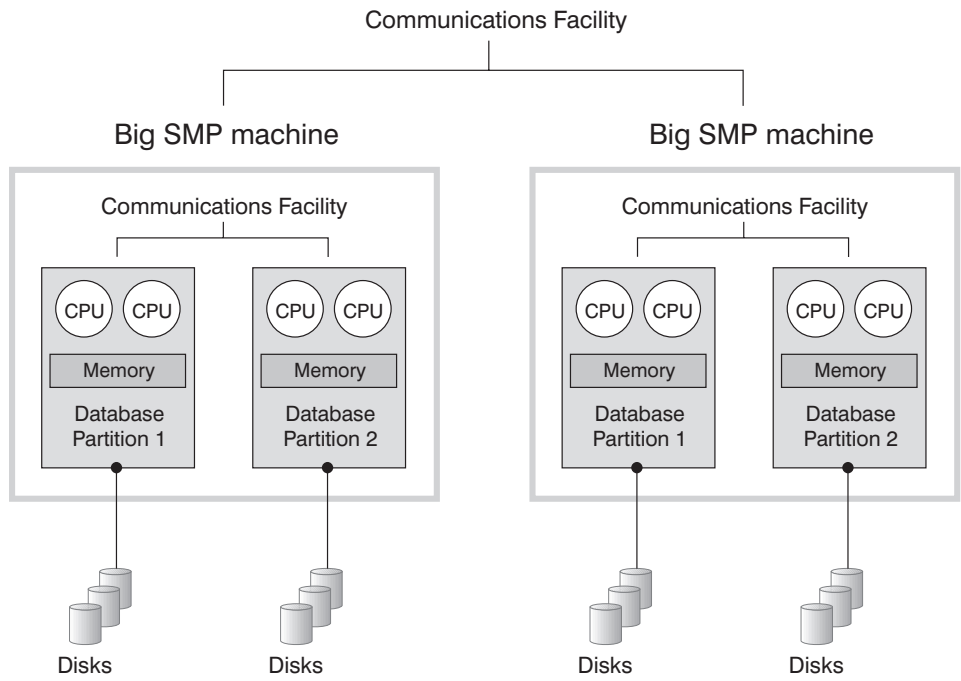


Figure 11. Partitioned Database, Symmetric Multiprocessor Systems Clustered Together

Note also that the ability to have two or more partitions coexist on the same machine (regardless of the number of processors) allows greater flexibility in designing high availability configurations and fallover strategies. See Appendix Q, “Supporting High Availability Cluster Multi-Processing Configurations” on page 865 for a description of how, upon machine failure, a database partition can be automatically moved and restarted on another machine already containing another partition of the same database.

Table 1 on page xli lists the types of parallelism best suited to take advantage of this hardware environment.

Summary of Parallelism Best Suited To Each Hardware Environment

The following table summarizes the types of parallelism best suited to the various hardware environments.

Table 1. Types of Parallelism Possible for Each Hardware Environment

Hardware Environment	I/O Parallelism	Intra-Query Parallelism	
		Intra-Partition Parallelism	Inter-Partition Parallelism
Single Partition, Single Processor	Yes	No(1)	No

Table 1. Types of Parallelism Possible for Each Hardware Environment			
Hardware Environment	**I/O Parallelism**	**Intra-Query Parallelism**	
		Intra-Partition Parallelism	**Inter-Partition Parallelism**
Single Partition, Multiple Processors (SMP)	Yes	Yes	No
Multiple Partitions, One Processor (MPP)	Yes	No(1)	Yes
Multiple Partitions, Multiple Processors (cluster of SMPs)	Yes	Yes	Yes
Logical Database Partitions	Yes	Yes	Yes
Note: (1) There may be an advantage to setting the degree of parallelism (using one of the configuration parameters) to greater than one even on a single CPU system, especially if the queries you execute are not fully utilizing the CPU (for example if they are I/O bound).			

Enabling Parallelism for Queries

There are two types of query parallelism: intra-partition parallelism and inter-partition parallelism. Either type, or both types, can be used depending on whether the environment is a single-partition or multi-partition environment.

Enabling Intra-Partition Query Parallelism

In order for intra-partition query parallelism to occur, you must modify database configuration parameters and database manager configuration parameters.

INTRA_PARALLEL
Database manager configuration parameter. See “Enable Intra-Partition Parallelism (intra_parallel)” on page 560 for more information.

DFT_DEGREE
Database configuration parameter. Provides the default for the DEGREE bind option and the CURRENT DEGREE special register. See “Default Degree (dft_degree)” on page 542 for more information.

DEGREE
Precompile or bind option for static SQL. See the *Command Reference* for more information.

CURRENT DEGREE
Special register for dynamic SQL. See the *SQL Reference* for more information.

For more information on the configuration parameter settings, and how to enable applications to process in parallel, see “Parallel Processing of Applications” on page 298.

Enabling Inter-Partition Query Parallelism

Inter-partition parallelism occurs automatically based on the number of database partitions and the distribution of data across these partitions.

Enabling Utility Parallelism

This section provides an overview of how to enable intra-partition parallelism for the following utilities:

- Load
- Create index
- Backup database / table space
- Restore database / table space

Inter-partition parallelism for utilities occurs automatically based on the number of database partitions.

Load

To enable parallelism while loading data, use the following parameters on the LOAD command:

- CPU_PARALLELISM
- DISK_PARALLELISM

See the *Command Reference* for information on the LOAD command.

Create Index

To enable parallelism when creating an index:

- The INTRA_PARALLEL database manager configuration parameter must be ON
- The table must be large enough to benefit from parallelism

See the *SQL Reference* for information on the CREATE INDEX statement.

Backup Database / Table Space

To enable parallelism when backing up a database or table space:

- The INTRA_PARALLEL database manager configuration parameter must be ON

To enable I/O parallelism when backing up a database or table space:

- Use more than one target media
- Configure table spaces for parallel I/O

See the *Command Reference* for information on the BACKUP DATABASE command.

Restore Database / Table Space

To enable parallelism when restoring a database or table space:

- The INTRA_PARALLEL parameter must be ON

To enable I/O parallelism when restoring a database or table space:

- Use more than one source media
- Configure table spaces for parallel I/O

See the *Command Reference* for information on the RESTORE DATABASE command.

Part 1. Database Design and Implementation

Chapter 1. Designing Your Logical Database

This section describes the following steps in database design:

- "Decide What Data to Record in the Database"
- "Define Tables for Each Type of Relationship" on page 5
- "Provide Column Definitions for All Tables" on page 7
- "Identify One or More Columns as a Primary Key" on page 9
- "Be Sure Equal Values Represent the Same Entity" on page 11
- "Consider Normalizing Your Tables" on page 11
- "Planning for Constraint Enforcement" on page 16
- "Other Database Design Considerations" on page 23.

Your goal in designing a database is to produce a representation of your environment that is easy to understand and will serve as a basis for expansion. In addition, you want a database design that will help you maintain consistency and integrity in your data. You can do this by producing a design that will reduce redundancy and eliminate anomalies that can occur during the updating of your database.

These steps are part of *logical* database design. Database design is not a linear process; you will probably have to redo steps as you work out the design.

The *physical* implementation of the database design is described in Chapter 2, "Designing Your Physical Database" on page 25 and Chapter 3, "Implementing Your Design" on page 55.

Decide What Data to Record in the Database

The first step in developing a database design is to identify the types of data to be stored in database tables. A database includes information about the *entities* in an organization or business and their relationships to each other. In a relational database, *entities* are defined as *tables*.

An *entity* is a person, object, or concept about which you wish to store information. Some of the entities described in the sample tables are employees, departments, and projects. (See Appendix H, "Sample Tables" on page 677, for a description of the sample database.)

In the sample employee table, the entity "employee" has *attributes*, or *properties*, such as employee number, name, work department, and salary amount. Those properties appear as the *columns* EMPNO, FIRSTNME, LASTNAME, WORKDEPT, 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."

Each row in a table represents an *occurrence* of an entity or relationship. For example, in the following table the values in the first row describe an employee named Haas.

Table 2. Occurrences of Employee Entities and their Attributes

EMPNO	FIRSTNME	LASTNAME	WORKDEPT	JOB
000010	Christine	Haas	A00	President
000020	Michael	Thompson	B01	Manager
000120	Sean	O'Connell	A00	Clerk
000130	Dolores	Quintana	C01	Analyst
000030	Sally	Kwan	C01	Manager
000140	Heather	Nicholls	C01	Analyst
000170	Masatoshi	Yoshimura	D11	Designer

There is a growing need to support non-traditional database applications such as multimedia. Within your design, you may want to consider attributes to support multimedia objects such as documents, video or mixed media, image, and voice.

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 perform a job

  Dolores works as an Analyst

- Departments report to other departments

  Department C01 reports to Department A00
  Department B01 reports to Department A00

- Employees work on projects

  Dolores and Heather both work on project IF1000

- Employees manage departments

  Sally manages department C01.

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.”

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 Sean O'Connell is a clerk in Department A00.

The information contained within a table depends on the relationships to be expressed, the amount of flexibility needed, and the data retrieval speed desired.

In addition to identifying data within your design, you should also identify other types of information such as the business rules which apply to that data.

Define Tables for Each Type of Relationship

In a database, you can express several types of relationships. Consider the possible relationships between employees and departments. An 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 *multi-valued* for departments. The relationship between employees (single-valued) and departments (multi-valued) 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 a 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.

The following topics are discussed within this section:

- "One-to-Many and Many-to-One Relationships"
- "Many-to-Many Relationships" on page 6
- "One-to-One Relationships" on page 7

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 the following example, the "many" side of the first and second relationships is "employees" so we define an employee table, EMPLOYEE.

Table 3. Many-to-One Relationships

Entity	Relationship	Entity
Employees	are assigned to	departments
Employees	work at	jobs
Departments	report to	(administrative) departments

In the third relationship, "departments" is the "many" side, so we define a department table, DEPARTMENT.

The following tables illustrate how these examples are represented:

The EMPLOYEE table:

EMPNO	WORKDEPT	JOB
000010	A00	President
000020	B01	Manager
000120	A00	Clerk
000130	C01	Analyst
000030	C01	Manager
000140	C01	Analyst
000170	D11	Designer

The DEPARTMENT table:

DEPTNO	ADMRDEPT
C01	A00
D01	A00
D11	D01

Figure 12. Assigning Many-to-One Facts to Tables

Many-to-Many Relationships

A relationship that is multi-valued 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. 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 the following example.

The following table illustrates how a many-to-many relationship (an employee can work on many projects and a project can have many employees working on it) can be represented:

The employee activity (EMP_ACT) table:

EMPNO	PROJNO
000030	IF1000
000030	IF2000
000130	IF1000
000140	IF2000
000250	AD3112

Figure 13. Assigning Many-to-Many Facts to Tables

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 the following example.

The following tables illustrates how a one-to-one relationship can be represented:

The DEPARTMENT table:

DEPTNO	MGRNO
A00	000010
B01	000020
D11	000060

Figure 14. Assigning One-to-One Facts to a Table

Provide Column Definitions for All Tables

To define a column in a relational 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 Appendix D, “Naming Rules” on page 629.

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

   The *data type* and *length* specify maximum length and the type of data that is valid for the column. Data types may be chosen from those provided by the database manager or you may choose to create your own user-defined types. For information about the data types provided by DB2 and about user-defined types, see the *SQL Reference* manual.

   Examples of data type categories are: numeric, character string, double-byte (or graphic) character string, date-time, and binary string.

   Large object (LOB) data types support multi-media objects such as documents, video, image and voice. These large objects are implemented using the following data types:

   - A *binary large object* (BLOB) string. Examples of BLOBs are photographs of employees, voice, and video.
   - A *character large object* (CLOB) string, where the sequence of characters can be either single- or multi-byte characters, or a combination of both. An example of a CLOB is a resume of an employee.
   - A *double-byte character for large object* (DBCLOB) string, where the sequence of characters are double-byte characters. An example of a DBCLOB is a Japanese resume.

   For a better understanding of large object support, refer to the *SQL Reference* manual.

   A *user-defined type* (UDT), is a type that is derived from an existing type. You may need to define types that are derived from existing types that share similar characteristics, but are considered to be separate and incompatible types.

   A *User-defined function* (UDF) may be used for a number of reasons, including invoking routines that allow comparison or conversion between user-defined types. UDFs extend and add to the support provided by built-in functions of SQL and can be used wherever a built-in function can be used. There are two types of UDFs:

   - An external function, which is written in a programming language
   - A sourced function, which will be used to invoke other UDFs

   For example, two numeric data types are European Shoe Size and American Shoe Size. Both types share the same representations of shoe size, but they are incompatible because the measurement base is different and cannot be compared. When this occurs, a user-defined function can be invoked to convert from one shoe size to another.

   During your design, you may have to consider functions for your UDTs. For a better understanding of user-defined types and user-defined functions, refer to the *SQL Reference* manual.

3. State 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 non-null default value to be assigned by the database manager or by the application.

Null values and default values are described in detail in the *SQL Reference* manual.

Identify One or More Columns as a Primary Key

The *unique key* of a table is a column or an ordered collection of columns for which each value identifies (functionally determines) a unique row. For example, an employee number column can be defined as a unique key, because each value in the column identifies only one employee. No two employees can have the same employee number.

The *primary key* of a table is one of the unique keys defined on a table but is selected to be the key of first importance on the table. There can only be one primary key on a table.

A *primary index* is automatically created for the primary key. The primary index is used by the database manager for efficient access to table rows and allows the database manager to enforce the uniqueness of the primary key. At other times the database manager may use other columns with indexes defined, and not only the primary key and index, to access data when processing queries.

Several columns could qualify as a candidate to be the primary key for a table. Each of the candidate columns could be considered unique. You could have all of the columns as part of the primary key but this would create an overly complex primary key. You should consider having just one of the columns as the primary key and then creating unique constraints or unique indexes on one or more of the other columns.

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 method used to distinguish unique rows.

Primary keys for some of the sample tables are:

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

The following example shows part of the project table with the primary key column indicated.

Table 4. A Primary Key on the PROJECT Table

PROJNO (Primary Key)	PROJNAME	DEPTNO
MA2100	Weld Line Automation	D01
MA2110	Weld Line Programming	D11

If every column in a table contains duplicate values, you cannot define a primary key with only one column. In this case, you can list two or more columns for the primary key. A key with more than one column is a *composite key*. The combination of column values should define a unique entity. If a composite key cannot be easily assigned, you may consider defining a new column that has unique values.

The following example shows a primary key containing more than one column; it is a *composite key*.

Table 5. A Composite Primary Key on the EMP_ACT Table

EMPNO (Primary Key)	PROJNO (Primary Key)	ACTNO (Primary Key)	EMPTIME	EMSTDATE (Primary Key)
000250	AD3112	60	1.0	1982-01-01
000250	AD3112	60	.5	1982-02-01
000250	AD3112	70	.5	1982-02-01

Identifying Candidate Key Columns

To identify candidate keys, select the smallest number of columns that define a unique entity. There may be more than one candidate key. In Table 19 on page 18, there appear to be many candidate keys. The EMPNO column, the PHONENO, and the LASTNAME each uniquely identify the employee.

The criteria for selecting a primary key from a pool of candidate keys should be persistence, uniqueness, and stability of the key.

- Persistence means that the primary key is always present for the row.
- Uniqueness means that each key value is and always will be different for each row.
- Stability means that the primary key should not be changed to another value.

Of the three candidate keys in the example, only the employee number meets the above criteria. An employee may not have a phone number when joining a company. Last names can change, and, although they are unique at one point, are not always guaranteed to be so. Therefore, the employee number column is the better choice for the primary key. An employee is assigned a unique number only once, and that number is generally not updated as long as the employee remains with the company. Since each employee must have a number, the employee number column is persistent.

Be Sure Equal Values Represent the Same Entity

You can have more than one table describing properties of the same set of entities. For example, the Employee Table shows the number of the department to which an employee is assigned, and the Department Table shows which manager is assigned to each department number. To retrieve both sets of properties simultaneously, you can join the two tables on the matching columns, as shown in the following example. The value in WORKDEPT and DEPTNO represent the same entity and represent a *join path* between the DEPARTMENT and EMPLOYEE tables.

The DEPARTMENT table:

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
D21	Administration Support	000070	D01

The EMPLOYEE table:

EMPNO	FIRSTNAME	LASTNAME	WORKDEPT	JOB
000250	Daniel	Smith	D21	Clerk

Figure 15. A Join Path between Two Tables

When you retrieve information about an entity from more than one table, make sure equal values represent the same entity. The connecting columns can have different names (like WORKDEPT and DEPTNO in the previous example), or they can have the same name (like the columns called DEPTNO in the department and project tables).

Consider Normalizing Your Tables

The topic of normalizing tables draws much attention in database design. Normalization helps you avoid redundancies and inconsistencies in your data. The main idea in normalization is to reduce tables to a set of columns where all the non-key columns depend on the entire primary key of the table. If this is not the case, the data can become inconsistent during updating.

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. (See “First Normal Form” on page 12)

Second Each column that is not in the key provides a fact that depends on the entire key. (See “Second Normal Form” on page 12)

Third Each non-key column provides a fact that is independent of other non-key columns and depends only on the key. (See “Third Normal Form” on page 14)

Fourth No row contains two or more independent multi-valued facts about an entity. (See “Fourth Normal Form” on page 15)

First Normal Form

A table satisfies the requirement of first normal form if for each row-and-column position in the table there exists one value, never a set of values. A table that is in first normal form does not necessarily meet the test for higher normal forms.

For example, the following table violates first normal form because the WAREHOUSE column contains several values for each occurrence of PART.

Table 6. Table Violating First Normal Form

PART (Primary Key)	WAREHOUSE
P0010	Warehouse A, Warehouse B, Warehouse C
P0020	Warehouse B, Warehouse D

The following example shows the table in first normal form.

Table 7. Table Conforms to First Normal Form

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY
P0010	Warehouse A	400
P0010	Warehouse B	543
P0010	Warehouse C	329
P0020	Warehouse B	200
P0020	Warehouse D	278

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.

This means that all data that is not part of the primary key must depend on all of the columns in the key. This reduces repetition among database tables.

Second normal form is violated when a non-key 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 the following example.

Table 8. Table Violates Second Normal Form

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY	WAREHOUSE_ADDRESS
P0010	Warehouse A	400	1608 New Field Road
P0010	Warehouse B	543	4141 Greenway Drive
P0010	Warehouse C	329	171 Pine Lane
P0020	Warehouse B	200	4141 Greenway Drive
P0020	Warehouse D	278	800 Massey Street

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, in Table 8, would be split into the following two tables:

Table 9. Part-Stock Table Conforms to Second Normal Form

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY
P0010	Warehouse A	400
P0010	Warehouse B	543
P0010	Warehouse C	329
P0020	Warehouse B	200
P0020	Warehouse D	278

Table 10. Warehouse Table Conforms to Second Normal Form

WAREHOUSE (Primary Key)	WAREHOUSE_ADDRESS
Warehouse A	1608 New Field Road
Warehouse B	4141 Greenway Drive
Warehouse C	171 Pine Lane
Warehouse D	800 Massey Street

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 Part 3, “Tuning Application Performance” on page 263.

Third Normal Form

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

Third normal form is violated when a non-key column is a fact about another non-key column. For example, the first table in the following example 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 the following example.

Table 11. Unnormalized Employee-Department Table Before Update

EMPNO (Primary Key)	FIRSTNAME	LASTNAME	WORKDEPT	DEPTNAME
000290	John	Parker	E11	Operations
000320	Ramlal	Mehta	E21	Software Support
000310	Maude	Setright	E11	Operations

The following example shows the content of the table following an update to the DEPTNAME column for John Parker. Note that there are now two different department names used for department number (WORKDEPT) E11:

Table 12. Unnormalized Employee-Department Table After Update. Information in table has become inconsistent.

EMPNO (Primary Key)	FIRSTNAME	LASTNAME	WORKDEPT	DEPTNAME
000290	John	Parker	E11	Installation Mgmt
000320	Ramlal	Mehta	E21	Software Support
000310	Maude	Setright	E11	Operations

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. The following tables are defined as a result of normalizing EMPDEPT.

Table 13. Employee Table After Normalizing the Employee-Department Table

EMPNO (Primary Key)	FIRSTNAME	LASTNAME	WORKDEPT
000290	John	Parker	E11
000320	Ramlal	Mehta	E21
000310	Maude	Setright	E11

Table 14. Department Table After Normalizing the Employee-Department Table

DEPTNO (Primary Key)	DEPTNAME
E11	Operations
E21	Software Support

Fourth Normal Form

A table is in fourth normal form if no row contains two or more independent multi-valued 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 following example:

Table 15. Table Violating Fourth Normal Form		
EMPNO (Primary Key)	**SKILL (Primary Key)**	**LANGUAGE (Primary Key)**
000130	Data Modelling	English
000130	Database Design	English
000130	Application Design	English
000130	Data Modelling	Spanish
000130	Database Design	Spanish
000130	Application Design	Spanish

Instead, the relationships should be represented in two tables, as in the following examples.

Table 16. Employee-Skill Table in Fourth Normal Form	
EMPNO (Primary Key)	**SKILL (Primary Key)**
000130	Data Modelling
000130	Database Design
000130	Application Design

Table 17. Employee-Language Table in Fourth Normal Form	
EMPNO (Primary Key)	**LANGUAGE (Primary Key)**
000130	English
000130	Spanish

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.

Planning for Constraint Enforcement

A *constraint* is a rule that the database manager enforces. Four types of constraint handling are covered in this section:

Unique Constraints Ensures the unique values of a key in a table. Any changes to the columns that compose the unique key are checked for uniqueness.

Referential Integrity	Enforces referential constraints on insert, update, and delete operations. It is the state of a database in which all values of all foreign keys are valid.
Table Check Constraints	Verify that changed data does not violate conditions specified when a table was created or altered.
Triggers	Define a set of actions that are executed when called by an update, delete, or insert operation on a specified table.

Unique Constraints

A *unique constraint* is the rule that the values of a key are valid only if they are unique within the table. Each column making up the key in a unique constraint must be defined as NOT NULL. Unique constraints are defined in the CREATE TABLE or the ALTER TABLE statements using the PRIMARY KEY clause or the UNIQUE clause.

A table can have any number of unique constraints; however, you can only define one unique constraint as the primary key for a table. Also, a table cannot have more than one unique constraint on the same set of columns.

When a unique constraint is defined, the database manager creates (if needed) a unique index and designates it as either a primary or unique system-required index. The enforcement of the constraint is through the unique index. Once a unique constraint has been established on a column, the check for uniqueness during multiple row updates is deferred until the end of the update.

A unique constraint can also be used as the parent key in a referential constraint.

Referential Integrity

Referential integrity lets you define required relationships between and within tables. The database manager maintains these relationships which are expressed as *referential constraints* and require that all values of a given attribute or column of a table also exist in some other table or column. For example, a typical referential constraint might require that every employee in the EMPLOYEE table must be in a department that exists in the DEPARTMENT table. No employee can be in a department that does not exist.

You can build referential constraints into a database to ensure that referential integrity is maintained. When planning for referential integrity, identify the relationships to be established between database tables. You can identify a relationship by defining a primary key and referential constraints.

The following two tables are related, and show some of the relationships to be discussed:

Table 18. DEPARTMENT Table

DEPTNO (Primary Key)	DEPTNAME	MGRNO
A00	Spiffy Computer Service Div.	000010
B01	Planning	000020
C01	Information Center	000030
D11	Manufacturing Systems	000060

Table 19. EMPLOYEE Table

EMPNO (Primary Key)	FIRSTNAME	LASTNAME	WORKDEPT (Foreign Key)	PHONENO
000010	Christine	Haas	A00	3978
000030	Sally	Kwan	C01	4738
000060	Irving	Stern	D11	6423
000120	Sean	O'Connell	A00	2167
000140	Heather	Nicholls	C01	1793
000170	Masatoshi	Yoshimura	D11	2890

The following definitions are useful for understanding referential integrity.

A *unique key* is a set of columns where no two values are duplicated in any other row. You may define one unique key for each table as the primary key. The unique key may also be known as a *parent key* when referenced by a foreign key.

A *primary key* is a unique key that is part of the definition of the table. Each table can only have one primary key. In the preceding tables DEPTNO and EMPNO are the primary keys of the DEPARTMENT and EMPLOYEE tables.

A *foreign key* is a column or set of columns in a table that refer to a unique key or primary key of the same or another table. A foreign key is used to establish a relationship with a unique key or primary key to enforce referential integrity among tables. The column WORKDEPT in the EMPLOYEE table is a foreign key because it refers to the primary key, column DEPTNO, in the DEPARTMENT table.

A *composite key* is a key that has more than one column. Unique primary and foreign keys can be composite keys. For example, if departments were uniquely identified by the combination of division number and department number, two columns would be needed to comprise the key to the DEPARTMENT table.

A *parent key* is a primary key or unique key of a referential constraint.

A *parent table* is a table containing a parent key that is related to at least one foreign key in the same or another table. A table can be a parent in an arbitrary number of relationships. For example, the DEPARTMENT table, which has a primary key of

DEPTNO, is a parent of the EMPLOYEE table, which contains the foreign key WORKDEPT.

A *parent row* is a row of a parent table whose parent key value matches at least one foreign key value in a dependent table. A row in a parent table is not necessarily a parent row. The fourth row (D11) of the DEPARTMENT table is the parent row of the third and sixth rows in the EMPLOYEE table. The second row (B01) of the DEPARTMENT table is not the parent of any other rows.

A *dependent table* is a table containing one or more foreign keys. A dependent table can also be a parent table. A table can be a dependent in an arbitrary number of relationships. For example, the EMPLOYEE table contains the foreign key WORKDEPT, which is dependent on the DEPARTMENT table that has a primary key.

A *dependent row* is a row of a dependent table that has a non-null foreign key value that matches a parent key value. The foreign key value represents a reference from the dependent row to the parent row. Since foreign keys may accept null values, a row in a dependent table is not necessarily a dependent row.

A table is a *descendent* of a table if it is a dependent table or if it is a descendent of a dependent table. A descendent table contains a foreign key that can be traced back to the parent key of some table.

A *referential cycle* is a path that connects a table to itself. When a table is directly connected to itself, it is a *self-referencing* table. If the EMPLOYEE table has another column called MGRID that contains the EMPNO of each employee's manager, then the EMPLOYEE table would be a self-referencing table. MGRID would be a foreign key for the EMPLOYEE table.

A *referential constraint* is an assertion that non-null values of a designated foreign key are valid only if they also appear as values of a unique key of a designated table. The purpose of referential constraints is to guarantee that database relationships are maintained and data entry rules are followed.

A *self-referencing* table is both a parent and a dependent in the same relationship. A self-referencing row is a row that is a parent and a dependent of itself. The constraint that exists in this situation is called a *self-referencing* constraint. For example, if the value of the foreign key in a row of a self-referencing table matches the value of the unique key in that row, then the row is self-referencing.

The following additional topic is discussed within this section:

- “Implications for SQL Operations”

Implications for SQL Operations

Enforcement of referential constraints has special implications for some SQL operations that depend on whether the table is a parent or a dependent. This segment describes the effects of referential integrity on the SQL INSERT, DELETE, UPDATE, and DROP operations.

The database manager does **not** automatically enforce referential constraints across systems. As a result, if you wish to enforce referential constraints across systems, your application programs must contain the necessary logic.

The following referential integrity rules are discussed:

- INSERT Rules
- DELETE Rules
- UPDATE Rules.

INSERT Rules: You can insert a row at any time into a parent table without any action being taken in the dependent table. However, 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 of the row that is being inserted, unless the foreign key value is null. The value of a composite foreign key is null if any component of the value is null.

This rule is implicit when a foreign key is specified.

When you try to insert a row into a table that has referential constraints, the INSERT operation is not allowed if any of the non-null foreign key values are not present in the parent key. If the INSERT operation fails for one row during an attempt to insert more than one row, all rows in the statement are backed out.

DELETE Rules: When you delete a row from a parent table, the database manager checks if there are any dependent rows in the dependent table with matching foreign key values. If any dependent rows are found, several actions could be taken. You can determine which action will be taken by specifying a *delete* rule when you create the dependent table.

The delete rules for a dependent table (the table containing the foreign key) when a primary key is deleted are:

RESTRICT — Prevents any row in the parent table from being deleted if any dependent rows are found. If you need to remove both parent and dependent rows, delete dependent rows first. Deleting the parent row first would violate the referential constraint and is not allowed.

See the *SQL Reference* for an example where this is different from NO ACTION.

NO ACTION — Enforces the presence of a parent row for every child after all the referential constraints are applied. See the *SQL Reference* for an example where this is different from RESTRICT.

CASCADE — Implies that deleting a row in the parent table automatically deletes any related rows in the dependent table. This rule is useful when a row in the dependent table has no significance without a row in the parent table.

Deleting the parent row first would automatically delete the dependent rows referencing a primary key. Therefore, the dependent rows would not need to be deleted first. If some of these dependent rows have dependents of their own, the delete rule for those relationships will be applied. In other words, the database manager can handle cascading deletions.

SET NULL Ensures that deletion of a row in the parent table sets the values of the foreign key in any dependent rows to null. Other parts of the row are unchanged.

If no delete rule is explicitly defined when the table is created, the NO ACTION rule will be applied.

Any table that can be involved in a delete operation is said to be delete-connected. The following restrictions apply to delete-connected relationships.

- A table cannot be delete-connected to itself in a referential cycle of more than one table.
- When a table is delete-connected to another table through more than one dependent relationship, these relationships must have the same delete rule, either CASCADE or NO ACTION.
- When a self-referencing table is a dependent of another table in a CASCADE relationship, the delete rule of the self-referencing relationship must also be CASCADE.

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 parent key of the employee table. If a manager retires, there is an effect on the department table.)

UPDATE Rules: The database manager prevents the update of a unique key of a parent row. When you update a foreign key in a dependent table, and the foreign key is not null, it must match some value of the parent key of the parent table of the relationship. If any referential constraint is violated by an UPDATE operation, an error occurs and no rows are updated.

When a value in a column of the parent key is updated:

- If any row in the dependent table matches the original value of the key, the update is rejected when the update rule is RESTRICT.
- If any row in the dependent table does not have a corresponding parent key when the update statement is completed (excluding after triggers), the update is rejected when the update rule is NO ACTION.

To update the value of a parent key that is in a parent row, you must first remove the relationship to any child rows in the dependent tables by either:

- Deleting the child rows; or,
- Update the foreign keys in dependent tables to include another valid key value.

When there is no dependency to the key value in the row, the row is no longer a parent in a referential relationship and can be updated.

If part of a foreign key is being updated and no part of the foreign key value is null, the new value of the foreign key must appear as a unique key value in the parent table. If there is no foreign key dependent on a given unique key, that is, the row containing the unique key is **not** a parent row, then part of the unique key may be updated. However, no more than one row can be selected for updating in this case, because you are working with a unique key where duplicate rows are not allowed.

Table Check Constraints

Business rules identified within your design can be enforced through table check constraints. *Table check constraints* specify search conditions that are enforced for each row of a table. These constraints are automatically activated when an update or insert statement runs against the table. They are defined when using either CREATE TABLE or ALTER TABLE statements.

A table check constraint can be used for validation. For example: the values of a department number must lie within the range 10 to 100; the job title of an employee can only be 'Sales', 'Manager', or 'Clerk'; or an employee who has been with the company for more than 8 years must earn more than $40,500.

See Chapter 5, “Utilities for Moving Data” on page 141 for more information on the impact of table check constraints on the IMPORT and LOAD commands.

Triggers

A *trigger* is a defined set of actions that are executed when a delete, insert, or update operation is carried out against a specified table. To help support business rules, triggers can be defined. Triggers are stored in the database, therefore application development is faster because you do not have to code the actions in every application program. The trigger is coded once, stored in the database and automatically called by the database manager, as required, when an application uses the database. This ensures that the business rules related to the data are always enforced. If a business rule does change, only a modification to the trigger is required instead of to each application program.

For example, triggers can be used to automatically update summary or audit data.

A user-defined function (UDF) can be called within a triggered SQL statement. This allows the triggered action to perform a non-SQL operation when the trigger is fired. For example, e-mail can be sent as an alert mechanism. For more information on triggers, see “Creating a Trigger” on page 89 and the *Embedded SQL Programming Guide* manual.

Other Database Design Considerations

When designing a database, it is important to consider which tables each user should be able to access. Access to tables is granted or revoked through authorizations. The highest level of authority is the system administration authority (SYSADM). A user with SYSADM authority can assign other authorizations, including the database administrator authority (DBADM).

There are other requirements that you may have to consider during your design, such as *audit*, *history*, *summary*, *security*, and *parallel processing capability*.

For *audit* purposes, you may have to record every update made to your data for a specified period. For example, you may want to update an audit table, each time an employee's salary was changed. Updates to this table could be made automatically if a trigger was established to enforce this behavior.

For performance reasons, you may only want to access a selected amount of data, while maintaining the base data as *history*. You should include within your design, the requirements for maintaining this historical data, such as the number of months or years of data that is required to be available before it can be purged.

There may be situations identified within your design that deal with *summary* information. For example, you may want to keep track of the number of active employees. In this case, a summary table could be updated each time a new employee joined the company, or when an existing employee left the company.

Security implications should also be identified within your design. For example, you may decide to support user access to certain types of data through security tables. You can define access levels to various types of data and who can access this data. Confidential data such as employee and payroll data, would have stringent security restrictions imposed where only a select number of individuals could be authorized to view this data, whereas certain time reporting data could be set up to be viewed globally. For more information on security and authorizations, see Chapter 4, “Controlling Database Access” on page 111.

As your business grows, you may need the additional capacity and performance capability provided by DB2 Extended Enterprise Edition. In this environment, your database is partitioned across several machines or systems, each responsible for the storage and retrieval of a portion of the overall database. In this environment, each partition (or node) of the database works in parallel to handle SQL or utility operations.

Issues and considerations relating to parallel operations are presented as appropriate to the topics presented in the following chapters. These issues and considerations are typically found toward the end of each topic.

Chapter 2. Designing Your Physical Database

After you have completed Chapter 1, Designing Your Logical Database and before Chapter 3, Implementing Your Design, there are a number of factors you should consider about the physical environment in which your database and tables will be implemented. These factors include understanding the files that will be created to support and manage your database, understanding how much space will be required to store your data, and determining how you should use table spaces that are required to store your data.

The following topics are discussed:

- Database Physical Directories
- Estimating Space Requirements for Tables
- Additional Space Requirements
- Designing Nodegroups
- Designing and Choosing Table Spaces

Database Physical Directories

When a database is created, the database manager creates a separate subdirectory to store control files (such as log header files) and to allocate containers to default table spaces. Objects associated with the database are not always stored in the database directory; they can be stored in various locations, including directly on devices.

The database is created in the instance that is defined in the DB2INSTANCE environment variable or in the instance to which you have explicitly attached (using the ATTACH command). See the “Using Multiple Instances of the Database Manager” on page 56 for an introduction to instances.

The naming scheme used on UNIX platforms is

```
current_path/$DB2INSTANCE/NODE/SQL00001
```

The naming scheme used on Intel platforms is

```
D:current_path\$DB2INSTANCE\NODE\SQL00001
```

where “D:” is a “drive letter” identifying the volume where the root directory is located.

`SQL00001` contains objects associated with the first database created, and subsequent databases are given higher numbers: `SQL00002` and so on.

The subdirectories are created in a directory with the same name as the database manager instance to which you are attached when you are creating the database. (On Intel platforms, the subdirectories are created under the root directory on a given volume which is identified by a “drive letter.”) These instance and database subdirectories are created within the path specified in the CREATE DATABASE command, and the database manager maintains them automatically. Depending on your platform, each instance might be owned by an instance owner, who has system administrator (SYSADM) authority over the databases belonging to that instance.

To avoid potential problems, do not create directories that use the same naming scheme, and do not manipulate directories that have already been created by the database manager.

Database Physical Files

The following files are found within the database:

File Name	Description
SQLDBCON	This file stores the tuning parameters and flags for the database. See Chapter 19, “Configuring DB2” on page 459 for information about changing database configuration parameters.
SQLOGCTL.LFH	This file is used to help track and control all of the database log files.
Syyyyyyy.LOG	Database log files, numbered from 0000000 to 9999999. The number of these files is controlled by the *logprimary* and *logsecond* configuration parameters. The size of the individual files is controlled by the *logfilsiz* configuration parameter. With circular logging, the files are reused and the same numbers will remain. With archival logging, the file numbers will increase in sequence as logs are archived and new logs are allocated. When 9999999 is reached, the number will wrap. By default, these log files are stored in a directory called SQLOGDIR. SQLOGDIR is found in the SQL*nnnnn* subdirectory.
SQLINSLK	This file is used to help ensure that a database is only used by one instance of the database manager.
SQLTMPLK	This file is used to help ensure that a database is only used by one instance of the database manager.
SQLSPCS.1	This file contains the definition and current state of all table spaces in the database.
SQLSPCS.2	This file is a copy of SQLSPCS.1, and is created for protection in case SQLSPCS.1 fails. Without one of these files, you will not be able to access your database.
SQLBP.1	This file contains the definition of all of the buffer pools used in the database.
SQLBP.2	This file is a copy of SQLBP.1 and is created for protection in case SQLBP.1 fails. Without one of these files, you will not be able to access your database.

Notes:

1. Do **not** make any direct changes to these files. They can only be accessed indirectly using the documented APIs and by tools that implement those APIs, including the command line processor commands and the graphical Control Center.
2. Do not remove these files.
3. Do not move these files.
4. The only supported means of backing up a database or table space is through the BACKUP API, including the command line processor and Control Center implementations of that API.

Estimating Space Requirements for Tables

The following information provides a general rule for estimating the size of a database:

- "System Catalog Tables"
- "User Table Data" on page 28
- "Long Field Data" on page 28
- "Large Object (LOB) Data" on page 29
- "Index Space" on page 30

After reading these sections, you should read "Designing and Choosing Table Spaces" on page 38.

Information is not provided for the space required by such things as:

- The local database directory file
- The system database directory file
- The file management overhead required by the operating system, including:
  - file block size
  - directory control space

Information such as row size and structure is precise. However, multiplication factors for file overhead because of disk fragmentation, free space, and variable length columns will vary in your own database since there is such a wide range of possibilities for the column types and lengths of rows in a database. After initially estimating your database size, create a test database and populate it with representative data. You will then find a multiplication factor that is more accurate for your own particular database design.

System Catalog Tables

When a database is initially created, system catalog tables are created. The system tables will grow as database objects and privileges are added to the database. Initially, they use approximately 1600 KB of disk space.

The amount of space allocated for the catalog tables depends on the type of table space and the extent size for the table space containing the catalog tables. For example, if a DMS table space with an extent size of 32 is used, the catalog table space will initially be allocated 20MB of space. For more information, see "Designing and Choosing Table Spaces" on page 38.

Note: For databases with multiple partitions, the catalog tables only reside on the partition where the CREATE DATABASE was issued. Disk space for the catalog tables is only required for that partition.

User Table Data

Table data is stored on 4KB pages. Each page contains 76 bytes of overhead for the database manager. This leaves 4020 bytes to hold user data (or rows), although no row can exceed 4005 bytes in length. A row will *not* span multiple pages.

Note that the table data pages **do not** contain the data for columns defined with LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, or DBCLOB data types. The rows in a table data page do, however, contain a descriptor of these columns. (See “Long Field Data” and “Large Object (LOB) Data” on page 29 for information about estimating the space required for the table objects that will contain the data stored using these data types.)

Rows are inserted into the table in a first-fit order. The file is searched (using a free space map) for the first available space that is large enough to hold the new row. When a row is updated, it is updated in place unless there is insufficient room left on the 4KB page to contain it. If this is the case, a record is created in the original row location which points to the new location in the table file of the updated row.

See “Long Field Data” and “Large Object (LOB) Data” on page 29 for information about how LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB and DBCLOB data is stored and for estimating the space required to store these types of columns.

For each user table in the database, the number of pages can be estimated by calculating:

```
ROUND DOWN(4020/(average row size + 8)) = records_per_page
```

Then use records_per_page with:

```
(number_of_records/records_per_page) * 1.1 = number_of_pages
```

The average row size is the sum of the average column sizes. For information on the size of each column, see CREATE TABLE in the *SQL Reference*.

The factor of “1.1” is for overhead such as page overhead and free space.

Long Field Data

If a table has LONG VARCHAR or LONG VARGRAPHIC data, in addition to the byte count of 20 for the LONG VARCHAR or LONG VARGRAPHIC descriptor (in the table row), the data itself must be stored. Long field data is stored in a separate table object which is structured differently from the other data types (see “User Table Data” and “Large Object (LOB) Data” on page 29).

Data is stored in 32KB areas that are broken up into segments whose sizes are “powers of two” times 512 bytes. (Hence these segments can be 512 bytes, 1024 bytes, 2048 bytes, and so on, up to 32,700 bytes.)

Each of these data types is stored in a fashion that enables free space to be reclaimed easily. Allocation and free space information is stored in 4KB allocation pages, which appear infrequently throughout the object.

The amount of unused space in the object depends on the size of the long field data and whether this size is relatively constant across all occurrences of the data. For data entries larger than 255 bytes, this unused space can be up to 50 percent of the size of the long field data.

If character data is less than 4KB in length, and it fits in the record with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of LONG VARCHAR or LONG VARGRAPHIC.

Large Object (LOB) Data

If a table has BLOB, CLOB, or DBCLOB data, in addition to the byte count (between 72 and 312 bytes) for the BLOB, CLOB, or DBCLOB descriptor (in the table row), the data itself must be stored. This data is stored in two separate table objects that are structured differently than other data types (see “User Table Data” on page 28 and “Long Field Data” on page 28).

To estimate the space required by large object data, you need to consider the two table objects used to store data defined with these data types:

- **LOB Data Objects**

  Data is stored in 64MB areas that are broken up into segments whose sizes are “powers of two” times 1024 bytes. (Hence these segments can be 1024 bytes, 2048 bytes, 4096 bytes, and so on, up to 64MB.)

  To reduce the amount of disk space used by the LOB data, you can use the COMPACT parameter on the *lob-options-clause* on the CREATE TABLE and ALTER TABLE statements. The COMPACT option minimizes the amount of disk space required by allowing the LOB data to be split into smaller segments so that it will use the smallest amount of space possible. This does not involve data compression but is simply using the minimum amount of space to the nearest 1KB boundary. Without the COMPACT option, there is no attempt to reduce the space used to the nearest 1KB boundary. Appending to LOB values stored using the COMPACT option may result in slower performance compared to appending LOB values for which the COMPACT option is not specified.

  The amount of free space contained in LOB data objects will be influenced by the amount of update and delete activity, as well as the size of the LOB values being inserted.

- **LOB Allocation Objects**

  Allocation and free space information is stored in 4KB allocation pages separated from the actual data. The number of these 4KB pages is dependent on the amount of data, including unused space, allocated for the large object data. The overhead is calculated as follows: one 4KB pages for every 64GB plus one 4KB page for every 8MB.

If character data is less than 4KB in length, and it fits in the record with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of BLOB, CLOB or DBCLOB.

Index Space

For each index, the space needed can be estimated as:

```
(average index key size + 8) * number of rows * 2
```

where:

- The “average index key size” is the byte count of each column in the index key. See the CREATE TABLE statement in the *SQL Reference* for information on how to calculate the byte count for columns with different data types. (Note that to estimate the average column size for VARCHAR and VARGRAPHIC columns, use an average of the current data size, plus one byte. Do not use the maximum declared size.)
- The factor of 2 is for overhead, such as non-leaf pages and free space.

Note: For every column that allows nulls, add one extra byte for the null indicator.

Temporary space is required when creating the index. The maximum amount of temporary space required during index creation can be estimated as:

```
(average index key size + 8) * number of rows * 3.2
```

where the factor of 3.2 is for index overhead as well as space required for the sorting needed to create the index.

Note: In the case of non-unique indexes, only four (4) bytes are required to store duplicate key entries. The estimates shown above assume no duplicates. The space required to store an index may be over-estimated by the formula shown above.

The following two calculations can be used to estimate the number of leaf pages. The results are not guaranteed. The results are only an estimate, and the accuracy depends largely on how well the averages used reflect the actual data.

Note: For SMS, the minimum space is 12KB. For DMS, the minimum is an extent.

The average number of keys per page is roughly:

```
(.9 * (3997 - (M*2))) * (D + 1)
-------------------------------
      K + 6 + (4 * D)
```

where:

- M = 3997 / (8 + minimumKeySize)
- D = average number of duplicates per key value
- K = averageKeySize

Remember that minimumKeySize and averageKeysize must have an extra 1 byte for each nullable key part.

The number of pages (including non-leaf pages) can be roughly calculated as follows:

```
P = number of pages = 0 initially
X = total number of pages
N = average number of keys per page
Y = X/N

While (Y > 1)
{
  P = P + Y
  Y = Y / N
}
P = P + P/16000 + 3
```

Finally, the amount of space for the index is P * 4096.

For DMS table spaces, add together the total sizes for all indexes on a table and round up to a multiple of the extent size for the table space where the index resides.

Additional Space Requirements

Additional space is also required as follows:

- "Log File Space"
- "Temporary Work Space" on page 32

Log File Space

The amount of space (in bytes) required for log files can range from:

```
( logprimary * (logfilsiz + 2 ) * 4096 ) + 8192
```

to:

```
( (logprimary + logsecond) * (logfilsiz + 2 ) * 4096 ) + 8192
```

where:

- *logprimary* is the number of primary log files as defined in the database configuration file (see "Number of Primary Log Files (logprimary)" on page 521)
- *logsecond* is the number of secondary log files as defined in the database configuration file (see "Number of Secondary Log Files (logsecond)" on page 522)
- *logfilsiz* is the number of pages in each log file as defined in the database configuration file (see "Size of Log Files (logfilsiz)" on page 519)
- 2 is the number of header pages required for each log file
- 4096 is the number of bytes in one page
- 8192 is the size (in bytes) of the log control file.

The upper limit of log space is dependent on the actual number of secondary log files that the database manager requires at run time. This upper limit may never be used or may only be used during occasional periods of high-volume activity.

Note: If the database is enabled for roll-forward recovery, special log space requirements should be considered:

- With the *logretain* configuration parameter enabled, the log files will be archived in the log path directory. The on-line disk space will eventually fill up, unless you move the log files to a different location.
- With the *userexit* configuration parameter enabled, a user exit program moves the archived log files to a different location. Extra log space is still required to allow for:
  - On-line archived logs that are waiting to be moved by the user exit program
  - New log files being formatted for future use.

Temporary Work Space

Some SQL statements require temporary tables for processing (such as a work file for sorts that cannot be done in memory). These require disk space for storage during the time they are used. The amount required will be totally dependent on the queries and the size of tables returned, and therefore cannot be estimated.

You can use the database system monitor and query table space APIs to help you observe the amount of work space being used during the normal course of operations.

Designing Nodegroups

A *nodegroup* is a named set of one or more nodes that are defined as belonging to a database. Each database partition that is part of the database system configuration must already be defined in a *partition configuration file* called *db2nodes.cfg*. A nodegroup can contain from one database partition to the entire number of database partitions defined for the database system.

You create a new nodegroup using the CREATE NODEGROUP statement. You modify a nodegroup using the ALTER NODEGROUP statement. You can add or drop one or more database partitions from a nodegroup. The database partitions must be defined in the *db2nodes.cfg* file before modifying the nodegroup. Table spaces (defined later) reside within nodegroups. Tables reside within table spaces.

When a nodegroup is created or modified, a partitioning map is associated with it. A partitioning map, in conjunction with a partitioning key and a hashing algorithm, is used by the database manager to determine which database partition in the nodegroup will store a given row of data. More information on partitioning maps, keys, and other related issues are discussed later in this chapter.

With a non-partitioned database, no partitioning key or partitioning map is required. There are no nodegroup design considerations if you are using a non-partitioned database. A *database partition* is part of the database that consists of its own user data, indexes, configuration files, and transaction logs. Default nodegroups that were created when the database was created, are used by the database manager. IBMCATGROUP is the default nodegroup for the table space containing the system catalogs. IBMTEMPGROUP is the default nodegroup for the table spaces containing the temporary tables. IBMDEFAULTGROUP is the default nodegroup for the table spaces containing the user-defined tables the user chooses to put there.

If you are using a multiple partition nodegroup, consider the following design points:

- In a multiple partition nodegroup, you can only create a unique index if it is a superset of the partitioning key.
- Depending on the number of database partitions in the database, you may have one or more single-partition nodegroups and one or more multiple partition nodegroups present.
- There can be no duplicate database partitions within a nodegroup, although the same database partition may be found in one or more nodegroups.
- To ensure fast recovery of the database partition with the system catalog tables, avoid placing user tables on the same database partition. This is accomplished by placing user tables in nodegroups that do not include the database partition in the IBMCATGROUP nodegroup.

You should place small tables in single database partition nodegroups, except where you want to take advantage of *collocation* with a larger table. Collocation is the placement of rows from different tables that contain related data in the same database partition. Collocated tables allow the database to utilize more efficient join strategies. Collocated tables can reside in a single database partition nodegroup. Tables are considered collocated if they reside in a multiple partition nodegroup, and have the same number of columns in the partitioning key and the data types of the corresponding columns are partition compatible. Rows in collocated tables with the same partitioning key value are placed on the same database partition. Tables can be in separate table spaces in the same nodegroup and still be considered collocated.

You should avoid extending medium-sized tables across too many database partitions. For example, a 100 MB table may perform better on a 16-database partition nodegroup than on a 32-database partition nodegroup.

You can use nodegroups to separate online-transaction-processing (OLTP) tables from decision-support tables to ensure that the performance of OLTP transactions is not impacted by decision-support transactions.

Nodegroup Design Considerations

Based on the logical design of your database, and the amount of data that the database is required to process, you should have a good idea whether your database needs to be partitioned. If you need to partition your database, you should consider the following to complete your database design as it relates to nodegroup use:

Data Partitioning

DB2 supports a partitioned storage model allowing you to store data across several database partitions in the database. This means that the data is physically stored across more than one database partition and yet can be accessed as if the data were

located in the same place. Applications and users accessing data in a partitioned database do not need to be aware of the location of the data.

The data, while physically split, is used and managed as a logical whole. Users can choose how to partition their data by declaring partitioning keys. Users can also determine which and how many database partitions their table data can be spread across by selecting the table space and the associated nodegroup in which the data should be stored. In addition, a partitioning map (which is user-updateable) is used with a hashing algorithm to specify the mapping of partitioning key values to database partitions which determines the placement and retrieval of each row of data. As a result, you can spread the workload across a partitioned database for large tables while allowing smaller tables to be stored on one or more database partitions. Each database partition has local indexes on the data it stores resulting in increased performance for local data access.

You are not restricted in your design to having all tables in their table spaces divided equally across all database partitions in the database. DB2 supports *partial declustering*, which means that you can divide tables and their table spaces across a subset of database partitions in the system (that is, a nodegroup). You do not have to divide all tables in their table spaces across all the database partitions in the system.

Partitioning Maps

In a partitioned database environment, the database manager has to have a way of knowing which rows of a table are stored on which database partition in the database. The database manager has to know where to go to look at or retrieve the data it needs. Just as we need a map to find our way around a city to different locations, the database manager needs a map, called a *partitioning map*, to find the right part of the database (that is, which database partition) to go to get different parts of the data in the database.

A partitioning map is an internally generated array containing either 4096 entries for multiple partition nodegroups, or a single entry for single partition nodegroups. For a single partition nodegroup, the partitioning map has only one entry containing the partition number of the database partition where all the rows of a database table are stored. For multiple partition nodegroups, the partition numbers of the nodegroup are specified in a round-robin fashion. Just as a city map is organized into sections using a grid, the database manager uses a *partitioning key* to determine the location (the database partition) where the data is stored.

For example, assume that you have a database created on four database partitions (numbered 0–3). The partitioning map for the IBMDEFAULTGROUP nodegroup of this database would be:

```
0 1 2 3 0 1 2 ...
```

If a nodegroup had been created in the database using database partitions 1 and 2, the partitioning map for that nodegroup would be:

```
1 2 1 2 1 2 1 ...
```

If the partitioning key for a table to be loaded in the database is an integer that has possible values between 1 and 500 000, the partitioning key is hashed to a partition number between 0 and 4 095. That number is used as an index into the partitioning map to select the database partition for that row.

Figure 16 shows how the row with the partitioning key value (c1, c2, c3) is mapped to partition 2, which, in turn, references database partition n5.

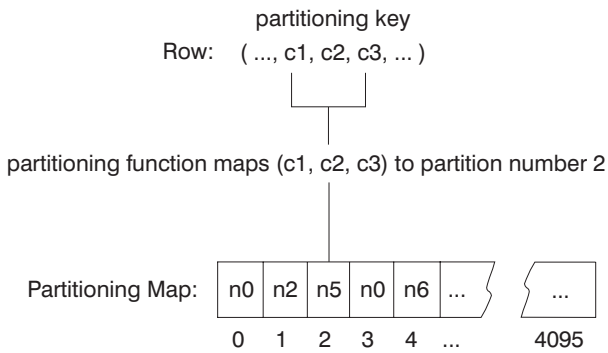


Figure 16. Data Distribution Using a Partition Map

A partition map is a flexible way of controlling where data is stored in a partitioned database. If you have a need at some future time to change the data distribution across the database partitions in your database, you can use the data redistribution utility. The data redistribution utility allows you to re-balance or introduce skew into the data distribution. For more information regarding this utility, see Chapter 16, “Redistributing Data Across Database Partitions” on page 435.

You can use the Get Table Partitioning Information (sqlugtpi) API to obtain a copy of a partitioning map that you can view. For more information on this API, see the *API Reference* manual.

Partitioning Keys

A *partitioning key* is a column (or group of columns) that is used to determine the partition in which a particular row of data is stored. A partitioning key is defined on a table using the CREATE TABLE statement. If a partitioning key is not defined for a table in a table space that is divided across more than one database partition in a nodegroup, one is created by default from the first column of the primary key. If no primary key is specified, the default partitioning key is the first non-long field column defined on that table. (*Long* includes all long data types and all Large Object data types). If you are creating a table in a table space associated with a single database partition nodegroup and you want to have a partitioning key, you must define the partitioning key explicitly. One is not created by default.

If no columns satisfy the requirement of the default partitioning key, the table is created without one. Tables without a partitioning key are only allowed in single database partition nodegroups. You can add or drop partitioning keys at a later time following the

initial creation of the table using the ALTER TABLE statement. Altering the partition key can only be done to a table in a table space that is associated with a single database partition nodegroup.

Choosing a good partitioning key is important. When you make the choice, you must know:

- How tables are to be accessed
- The nature of the query workload
- The join strategies employed by the database system.

If collocation is not a major consideration, a good partitioning key for a table is one that spreads the data evenly on all database partitions in the nodegroup. The partitioning key for each table in a table space that is associated with a nodegroup determines if the tables are collocated. Tables are considered collocated when:

- The tables are placed in table spaces that are in the same nodegroup
- The partition keys in each table have the same number of columns
- The data types of the corresponding columns are partition-compatible.

This ensures that rows of collocated tables with the same partitioning key values are located on the same partition. For more information on partition-compatibility, see “Partition Compatibility” on page 38. For more information on table collocation, see “Table Collocation” on page 37.

An inappropriate partitioning key can cause the distribution in the data of the table to be uneven. Columns with unevenly distributed data and columns with a small number of distinct values should not be chosen as a partitioning key. The number of distinct values must be great enough to ensure an even distribution of rows across all database partitions in the nodegroup. The cost of applying the partitioning hash algorithm is proportional to the size of the partitioning key. The partitioning key cannot be more than 16 columns, but fewer columns make for better performance. Unnecessary columns should not be included in the partitioning key.

The following points should be considered when defining partitioning keys:

- Creation of a table with only long data types (LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, and DBCLOB) is not allowed for multi-partition tables.
- Once defined, alteration of the partition key definition is not allowed.
- You cannot update the partitioning key column value for a row in the table.
- You can only delete or insert partitioning key column values.
- The partitioning key should include the most frequently joined columns.
- The partitioning key should be made up of columns that often participate in a GROUP BY clause.
- Any unique key or primary key must contain all the partitioning key columns.

- In an online-transaction processing (OLTP) environment, all columns in the partitioning key should participate in the transaction by using equal (=) predicates with constants or host variables. For example, assume you have an employee number, *emp_no* that is often used in transactions such as:

```
UPDATE emp_table SET ... WHERE
emp_no = host-variable
```

In a situation like this, the *emp_no* column is a good choice as a single column partitioning key for the *emp_table* table.

Hash partitioning is the method whereby the placement of each row in the partitioned table is determined. The method works as follows:

1. The hashing algorithm is applied to the value of the partitioning key.
2. The hashing algorithm generates a partitioning map number between zero (0) and 4095.
3. The partitioning map is created when a nodegroup is created. Each of the partition numbers is sequentially repeated in a round-robin fashion to fill the partition map. For more information on partitioning maps, see “Partitioning Maps” on page 34.
4. The partition map number is used as an index into the partitioning map. The number at that location in the partitioning map is the number of the database partition where the row is stored.

Table Collocation

When logically designing your database, and based on the needs of your applications, you may find that two or more tables will jointly provide data in response to frequently asked queries. When physically designing your database, you want related data from these two tables to be located as close together as possible. In an environment where the database is physically divided among two or more database partitions, there must be a way to keep the related pieces of the divided tables as close together as possible. The ability to do this is called *table collocation*. Tables are collocated when they are stored in the same nodegroup, and when their partitioning keys are compatible.

DB2 has the ability to recognize, when accessing more than one table for a join or subquery, that the data to be joined is located at the same database partition. When this happens, DB2 can choose to perform the join or subquery at the database partition where the data is stored instead of having to move data between database partitions. This ability to carry out joins or subqueries at the database partition has significant performance advantages. The data from the tables must be divided in the same way and then positioned or located at the same database partition.

Since both tables are to be divided across multiple database partitions, both tables require a partitioning key. Placing both tables in the same nodegroup ensures a common partition map. The tables may be in different table spaces, but the table spaces must be associated with the same nodegroup. The data types of the corresponding columns in each partition key must be *partition-compatible*.

Partition Compatibility

The base data types of corresponding columns of partitioning keys are compared and can be declared as being *partition compatible*. Partition compatible data types have the property that two variables, one of each type, with the same value, are mapped to the same partition number by the same partitioning algorithm.

Partition compatibility has the following characteristics:

- A base data type is compatible with another of the same base data type.
- Internal formats are used for DATE, TIME, and TIMESTAMP data types. They are not compatible with each other, and none are compatible with CHAR.
- Partition compatibility is not affected by columns with NOT NULL or FOR BIT DATA definitions.
- NULL values of compatible data types are treated identically. Different results might be produced for NULL values of non-compatible data types.
- Base data types of a User Defined Type are used to analyze partition compatibility.
- Decimals of the same value in the partitioning key are treated identically, even if their scale and precision differ.
- Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC) are ignored by the system-provided hashing algorithm.
- SMALLINT and INTEGER are compatible data types.
- REAL and FLOAT are compatible data types.
- CHAR and VARCHAR of different lengths are compatible data types.
- GRAPHIC and VARGRAPHIC are compatible data types.
- LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBLOB and BLOB data types are not applicable for partition compatibility since they are not supported as partitioning keys.

Designing and Choosing Table Spaces

A table space is a storage model that provides a level of indirection between a database and the tables stored within that database. Table spaces reside in nodegroups. Table spaces allow you to assign the location of database and table data directly onto containers. (A container can be a directory name, a device name, or a file name.) This can provide improved performance, more flexible configuration, and better integrity.

Since table spaces reside in nodegroups, the table space selected to hold a table defines how the data for the table is partitioned across the database partitions in a nodegroup. A single table space can span several containers. It is possible for multiple containers (from one or more table spaces) to be created on the same physical disk (or drive, in Intel terms). For improved performance, each container should use a different disk. The following diagram shows an example of the relationship between tables and table spaces within a database and the containers and disks associated with the database.

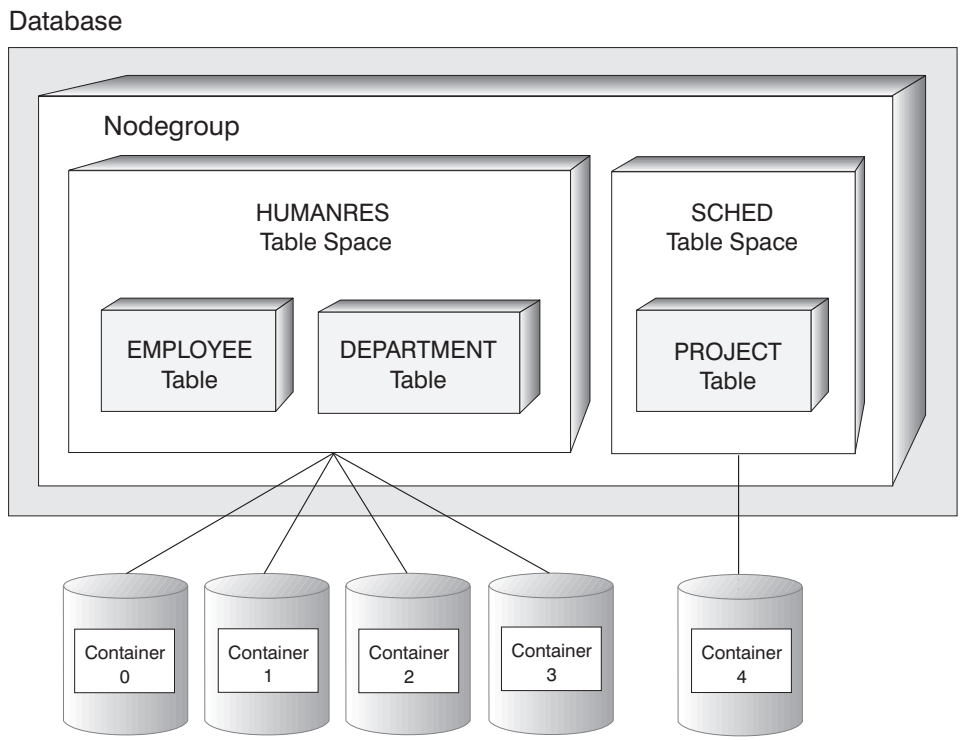


Figure 17. Table Spaces and Tables Within a Database

The EMPLOYEE and DEPARTMENT tables are in the HUMANRES table space which spans Containers 0, 1, 2 and 3. The PROJECT table is in the SCHED table space in Container 4. This example shows each container existing on a separate disk.

The database manager attempts to balance the load of the data across the containers. As a result, all containers will be used to store data. The number of pages that the database manager writes to a container before using a different container is called the *extent size*. The database manager does not always start storing table data in the first container.

The following diagram shows the HUMANRES table space with an extent size of two 4KB pages, and with four containers each with a small number of allocated extents. The DEPARTMENT and EMPLOYEE tables both have 7 pages and span all four containers.

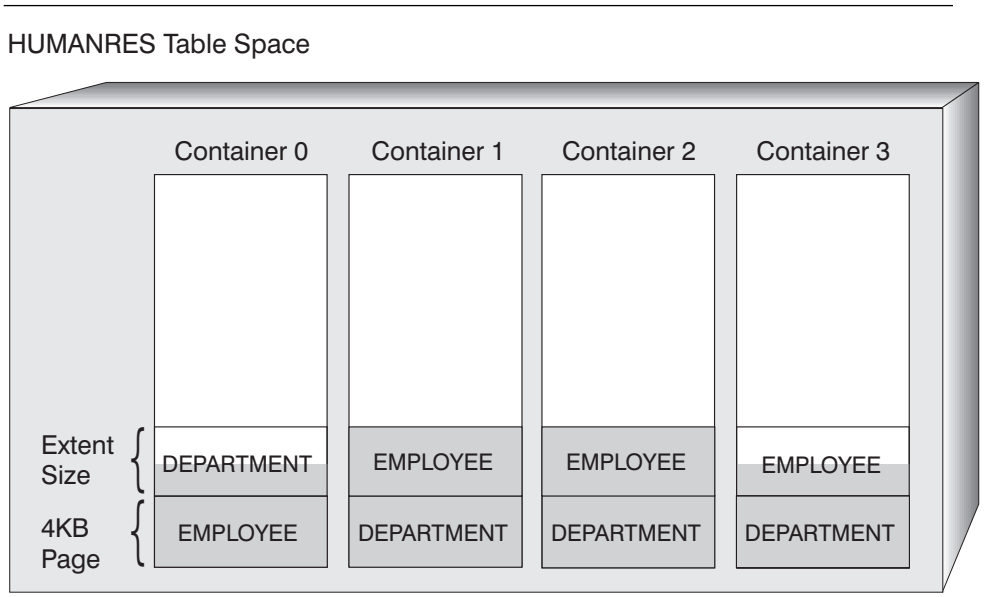


Figure 18. Use of Container and Extents

A database must contain at least three table spaces:

- One **catalog** table space, which contains all the system catalog tables for the database. This table space is called SYSCATSPACE and it cannot be dropped. IBMCATGROUP is the default nodegroup for this table space.
- One or more **user** table spaces, which contain all user-defined tables. By default, one table space, USERSPACE1, is created. IBMDEFAULTGROUP is the default nodegroup for this table space.

  You should specify a table space name when you create a table, or the results may not be what you intend. If you do not specify a table space name, the table is placed according to the following rules: if the table space IBMDEFAULTGROUP exists, then use it. If it does not exist, use a table space created by you if one exists. Otherwise, use USERSPACE1 if it exists. If USERSPACE1 does not exist, table creation fails.
- One or more **temporary** table spaces, which contain temporary tables. By default one table space called TEMPSPACE1 is created. A database must have at least one temporary table space. IBMTEMPGROUP is the default nodegroup for this table space.

  If a database uses more than one temporary table space, temporary objects are allocated among the temporary table spaces in a round robin fashion.

  An application may encounter a temp-tablespace-full condition when one of the table spaces is full even if there is still room in the other temporary table spaces. Thus, you should observe the following guidelines when creating temporary table spaces:

– Create as few temporary table spaces as possible. It's always better to put all of the containers into a single table space rather than dividing them up among multiple table spaces.
– Make all temporary table spaces the same size if possible.

Note: In a partitioned database environment, the catalog node will have all three table spaces and the other database partitions will each have only TEMPSPACE1 and USERSPACE1.

There are two types of table spaces, both of which can be used in a single database:

- System Managed Space Table Space: The operating system's file manager controls the storage space.
- Database Managed Space Table Space: The database manager controls the storage space.

After understanding the differences between these two types of table spaces, see "Table Space Design Considerations" on page 46.

System Managed Space Table Space

In a System Managed Space (SMS) table space, the operating system's file system manager allocates and manages the space where the table is to be stored. The storage model typically consists of many files, representing table objects, stored in the file system space. The user decides on the location of the files, DB2 controls their names, and the file system is responsible for managing them. By controlling the amount of data written to each file, the database manager evenly spreads the data over the table space containers. An SMS table space is the default table space.

In addition to the database physical files, each table has at least one SMS physical file associated with it. See "SMS Physical Files" on page 43 for a list of these files and a description of their contents.

In an SMS table space, the file is extended one page at a time as the object grows. When inserting a large number of rows, some delay may result from waiting for the system to allocate another page.

Note: If you need improved insert performance, you can consider enabling multipage file allocation. This allows the system to allocate or extend the file by more than one page at a time. You must run `db2empfa` to enable multipage file allocation. The db2empfa utility must be run on each database partition in a partitioned database. Once multipage file allocation is enabled, it cannot be disabled. See the *Command Reference* for more information on `db2empfa`.

You should explicitly define SMS table spaces using the MANAGED BY SYSTEM on the CREATE DATABASE command or on the CREATE TABLESPACE statement. You must consider two key factors when you design your SMS table spaces:

1. Containers for the table space

   You must specify the number of containers that you wish to use for your table space. It is very important to identify all the containers you want to use, since you

cannot add or delete containers after an SMS table space is created. In a partitioned database environment, when a new partition is added to the nodegroup for an SMS table space, the ALTER TABLESPACE statement can be used to add containers for the new partition.

Each container used for an SMS table space identifies an absolute or relative directory name. Each of these directories can be located on a different file system (or physical disk). As a result, the maximum size of the table space can be calculated by:

```
number of containers * (maximum file system size supported by the
operating system)
```

Note: This formula assumes that there is a distinct file system mapped to each container, and that each file system has the supported maximum of space available. In practice, this may not be the case and the practical maximum database size may be much smaller.

2. Extent size for the table space

   Similar to specifying the number of containers, the extent size can only be specified when the table space is created. Because it cannot be changed later, it is important to select an appropriate value for the extent size. See “Choosing an Extent Size” on page 50 for more information.

   When creating a table space, if you do not specify the extent size, the database manager will create the table space using the default extent size, defined by the *dft_extent_sz* database configuration parameter (see “Default Extent Size of Table Spaces (dft_extent_sz)” on page 506). This configuration parameter is initially set based on information provided when the database is created. If the DFT_EXTENTSIZE parameter is not specified on the CREATE DATABASE command, the default extent size will be set to 32.

To choose the appropriate values for the number of containers and the extent size for the table space, you must understand:

- The limitation that your operating system imposes on the size of a logical file system.

  For example, some operating systems have a 2GB limit. Therefore, if you want a 64GB table object, you will need at least 32 containers on this type of system.

  Check the limitations on size and the number of containers on the platform where you are working as part of your determination regarding the number of containers and the extent size for the table space.

  When you create the table space, you can specify containers that reside on different files systems and as a result increase the amount of data that can be stored in the database.

- How the database manager manages the data files and containers associated with a table space.

  The first table data file (SQL00001.DAT) is created in the first container specified for the table space, and this file is allowed to grow to the extent size. After it

reaches this size, the database manager writes the data to SQL00001.DAT in the next container. This process continues until all of the containers contain SQL00001.DAT files, at which time, the database manager returns to the first container to which data was written for that table. This process (known as *striping*) continues through the container directories until either a container becomes full at which time a -289 error is returned; or, no more space can be allocated from the operating system at which time a disk-full error is returned. This mechanism is also used for index (SQLnnnnn.INX), long field (SQLnnnnn.LF), and LOB (SQLnnnnn.LB and SQLnnnnn.LBA) files.

Note: The SMS table space is full as soon as any one of its containers is full. Thus, it is important to allocate the same amount of space for each container.

To help spread data across the containers more evenly, the database manager determines the container to start writing a table's data by taking the table's ID (1 in the above example) modulo the number of containers. Containers are numbered sequentially starting at 0.

See "SMS Physical Files" for more information about the files used in an SMS table space.

SMS Physical Files

The following files are found within an SMS table space directory container:

File Name	Description
SQLTAG.NAM	There is one of these files in each container subdirectory, and they are used by the database manager when you connect to the database to verify that the database is complete and consistent.
SQLxxxxx.DAT	Table file. All rows of a table are stored here, with the exception of LONG VARCHAR, LONG VARGRAPHIC, CLOB, BLOB or DBCLOB data.
SQLxxxxx.LF	File containing LONG VARCHAR or LONG VARGRAPHIC data (also called "long field data"). This file is only created if LONG VARCHAR or LONG VARGRAPHIC columns exist in the table.
SQLxxxxx.LB	Files containing BLOB, CLOB, or DBCLOB data (also called "LOB data"). These files are only created if BLOB, CLOB, or DBCLOB columns exist in the table.
SQLxxxxx.LBA	Files containing allocation and free space information about the SQLxxxxx.LB files.
SQLxxxxx.INX	Index file for a table. All indexes for the corresponding table are stored in this single file. It is only created if indexes have been defined. **Note:** When an index is dropped, the space is not physically freed from the index (.INX) file until the index file is deleted. The index file will be deleted if all the indexes on the table are dropped (and committed) or if the table is

reorganized. If the index file is not deleted, the space will be marked free once the drop has been committed, and will be reused for future index creations or index maintenance.

SQLxxxxx.DTR Temporary data file for a REORG of a DAT file. While reorganizing a table, the REORG utility creates a table in one of the temporary table spaces. These temporary table spaces can be defined to use containers different from those used for the user-defined tables.

SQLxxxxx.LFR Temporary data file for a REORG of a LF file. Notes for the .DTR file apply here as well.

SQLxxxxx.RLB Temporary data file for a REORG of a LB file. Notes for the .DTR file apply here as well.

SQLxxxxx.RBA Temporary data file for a REORG of a LBA file. Notes for the .DTR file apply here as well.

Notes:

1. Do **not** make any direct changes to these files. They can only be accessed indirectly using the documented APIs and by tools that implement those APIs, including the command line processor commands and the graphical Control Center.
2. Do not remove these files.
3. Do not move these files.
4. The only supported means of backing up a database or table space is through the BACKUP API, including implementations of that API, such as those provided by the command line processor and Control Center.

Database Managed Space Table Space

In a Database Managed Space (DMS) table space, the database manager controls the storage space. The storage model consists of a limited number of devices, whose space is managed by DB2. The Administrator decides which devices to use, and DB2 manages the space on the devices. This table space is essentially an implementation of a special purpose file system designed to best meet the needs of the database manager. The table space definition includes a list of the devices or files belonging to the table space in which data can be stored.

A DMS table space containing user-defined tables and data can be defined as:

- A **regular** table space to store normal table and index data
- A **long** table space to store long field or LOB data

When designing your DMS table spaces and containers, you should consider the following:

- The database manager uses striping to ensure an even distribution of data across all containers.

- The maximum size of the different types of table spaces:
  - Regular table and index data—64GB
  - Long field data—2TB
  - Temp data—2TB
- Unlike SMS table spaces, the containers that make up a DMS table space do not need to be the same size. Also, if any container is full, DMS table spaces use any available free space from other containers.
- The space is preallocated.

  Because it is preallocated, the space must be available before the table space can be created. When using device containers, the device must also exist with enough space for the definition of the container. Each device can have only one container defined to it, so to avoid wasted space, the size of the device and the size of the container should be equivalent. If, for example, the device is allocated with 5000 pages and the device container is defined to allocate 3000 pages, then 2000 pages on the device will not be usable.
- One page in every container is reserved for overhead and the remaining pages will be used one extent at a time. Only full extents are used in the container, so for optimal space management, you can use the following formula to help you determine the appropriate size to use when allocating a container:

  ```
  (extent size * n) + 1
  ```

  where, `extent size` is the size of each extent for the table space and `n` is the number of extents you want to store in the container.
- The number of extents you require:
  - Three extents in the table space are reserved for overhead
  - At least two extents are required to store any user table data. (These two extents allow for the regular data for one table, not for any index, long field or large object data which require their own extents.)
- Device containers must use logical volumes with a “character special interface,” not physical volumes.
- You can use files instead of devices with DMS table spaces. No operational difference exists between a file and a device; however, a file can be less efficient because of the runtime overhead associated with the filesystem. Files are useful when:
  - Devices are not directly supported
  - A device is not available
  - Maximum performance is not required
  - You do not want to set up devices.
- Some operating systems allow you to have physical devices greater than 2GB in size. You should consider partitioning the physical device into multiple logical devices so that no container is bigger than the size allowed by the operating system.

Adding Containers to DMS Table Spaces

You can add a container to an existing table space to increase its storage capacity with the ALTER TABLESPACE statement. The contents of the table space are then re-balanced across all containers. Access to the table space is not restricted during the re-balancing. If you need to add more than one container, you should add them at the same time either in one ALTER TABLESPACE statement or within the same transaction to prevent the database manager from having to re-balance the containers more than once.

You should check how full the containers for a table space are by using the LIST TABLESPACE CONTAINERS or the LIST TABLESPACES commands. Adding new containers should be done before the existing containers are almost or completely full. The new space across all the containers is not available until the re-balance is complete.

Adding a container which is smaller than existing containers results in a uneven distribution of data. This can cause parallel I/O operations, such as prefetching data, to perform less efficiently than they otherwise could on containers of equal size.

Table Space Design Considerations

Based on the logical design of your database, you should have a good idea of the size of each table, and as a result, of your database. Based on your understanding of this information, you should consider the following to complete your database design as it relates to table space use:

- Considerations for Table Space Input and Output (I/O)
- Mapping Table Spaces to Buffer Pools
- Mapping Table Spaces to Nodegroups
- Mapping Tables to Table Spaces
- Choosing an Extent Size
- Recommendations for Catalog and Temporary Table Spaces
- Workload Considerations
- Choosing an SMS or DMS Table Space

Considerations for Table Space Input and Output (I/O)

The type and design of your table space determines the efficiency of the I/O performed against that table space. Here are some concepts that you should understand before considering further the issues surrounding table space design and use.

Big-block reads

A read where several pages (usually an extent) is retrieved in a single request. Reading several pages at once is more efficient than reading each page separately.

Prefetching

The reading of pages in advance of those pages being referenced by a query. The overall objective is to reduce response time. This can be achieved if the prefetching of pages can occur asynchronously to the execution of the query. The best response time is achieved when

either the CPU(s) or the I/O subsystem are operating at maximum capacity.

Page cleaning

As pages are read and modified, these pages accumulate in the database buffer pool. Whenever a page is read in, there must be a buffer pool page to read it into. If the buffer pool is full of modified pages, one of these modified pages must be written out to the disk before the new page can be read in. To prevent the buffer pool from becoming full, page cleaner tasks write out modified pages in order to guarantee the availability of buffer pool pages for use by read requests.

Whenever it is advantageous, DB2 performs big-block reads. This typically occurs when retrieving data that is sequential or partially sequential in nature. The amount of data read in one read depends on the extent size -- the bigger the extent size, the more pages that are read at one time.

How the extent is stored on disk affects the I/O efficiency. When considering a DMS table space using device containers, the data tends to be contiguous on disk and can be read with a minimum of seek time and disk latency. However, if files are being used, the data may have been broken up by the file system and stored in more than one location on disk. This occurs most often when using SMS table spaces where files are extended one page at a time, making fragmentation more likely. Preallocation of a large file for use by a DMS table space tends to be contiguous on disk, especially if the file was allocated in a clean file space.

DB2 performing big-block reads is only one way in which query execution is assisted. You can control how aggressive prefetching can be by tuning the PREFETCHSIZE parameter on the CREATE TABLESPACE statement. (The default value for all table spaces in the database is set by the *dft_prefetch_sz* configuration parameter.) The PREFETCHSIZE parameter tells DB2 how many pages to read whenever a prefetch is triggered. By setting PREFETCHSIZE to a multiple of the EXTENTSIZE parameter on the CREATE TABLESPACE statement, you can cause multiple extents to be read in parallel. (The default value for all table spaces in the database is set by the *dft_extent_sz* configuration parameter. The EXTENTSIZE parameter specifies the number of 4K pages that will be written to a container before skipping to the next container.)

For example, suppose you had a table space that used three devices. If you set the PREFETCHSIZE to be three times the EXTENTSIZE, then DB2 can do a big-block read from each device in parallel, thereby significantly increasing the I/O throughput. This assumes that each device is a separate physical device and that the controller has sufficient bandwidth to handle the data stream from each device. Note that DB2 may have to dynamically adjust the prefetch parameters at runtime based on query speed, buffer pool utilization, and other factors.

You should know that some file systems use their own prefetching (such as the Journaled File System on AIX). In some cases, the file system prefetching is set to be

more aggressive than the DB2 prefetching. This results in situations where you observe that prefetching for SMS and DMS table spaces with file containers is outperforming prefetching for DMS table spaces with devices. This is misleading since it is likely the result of the additional level of prefetching that is occurring in the file system. DMS table spaces should be able to outperform any equivalent configuration.

For prefetching or even reading to be efficient, a sufficient number of clean buffer pool pages must exist into which to read the data. For example, there could be a parallel prefetch request which reads three extents from a table space and where a modified page must be written out from the buffer pool for each page being read. With the potential for a buffer page to be written out for every page being read in, it is clear that the prefetch request is slowed significantly perhaps to the point where it cannot keep up with the query. Page cleaners should be configured in sufficient numbers to satisfy the prefetch request. At least one page cleaner should be defined for each real disk used by the database. For more information on these topics and performance, see the Chapter 14, “Operational Performance” on page 395.

Mapping Table Spaces to Buffer Pools

Each table space is associated with a specific buffer pool. The default buffer pool is IBMDEFAULTBP. If another buffer pool is to be associated with a table space, the buffer pool must exist (it is defined with the CREATE BUFFERPOOL statement), and the association is defined when the table space is created (using the CREATE TABLESPACE statement). The association between the table space and the buffer pool can be changed using the ALTER TABLESPACE statement.

Having more than one buffer pool allows you to configure the memory used by the database to improve overall performance and to help with setting performance goals for specific applications. For example, for table spaces with one or more large tables which are accessed randomly by users, the size of the buffer pool can be limited since caching the data pages might not be beneficial. Another example would have the table space for an important online transaction application associated with a buffer pool that is larger than others. In this way, the data pages used by the application could be cached longer in the buffer pool resulting in lower response times. Care must be taken in configuring new buffer pools beyond the default.

The storage required for all the buffer pools must be available to the database manager when starting up the database. If DB2 is unable to obtain the storage required for all defined buffer pools, the database manager will start up with the default buffer pool of a minimal size, and issue a warning message.

In a partitioned database environment, you can create a buffer pool of the same size for all partitions in the database. You can also create buffer pools of particular sizes on different partitions. For more information on the CREATE BUFFERPOOL statement, see the *SQL Reference* manual.

Mapping Table Spaces to Nodegroups

In a partitioned database environment, each table space is associated with a specific nodegroup. This allows for the characteristics of the table space to be applied to each

node in the nodegroup. The nodegroup must exist (it is defined with the CREATE NODEGROUP statement), and the association between the table space and the nodegroup is defined when the table space is created using the CREATE TABLESPACE statement.

You cannot change the association between table space and nodegroup using the ALTER TABLESPACE statement. You can only change the table space specification for individual partitions within the nodegroup. If not in a partitioned database environment, each table space is associated with a default nodegroup. The default nodegroup when defining a table space is IBMDEFAULTGROUP unless a temporary table space is being defined and then IBMTEMPGROUP is used. For more information on the CREATE NODEGROUP statement, see the *SQL Reference* manual. For more information on nodegroups and physical database design, see the "Designing Nodegroups" on page 32.

Mapping Tables to Table Spaces

When determining how to map tables to table spaces in your design, you should consider:

- The partitioning of your tables.

  At a minimum, you should ensure that the table space you choose is in the nodegroup with the partitioning you desire.

- The amount of data in the table.

  If you plan to store many small tables in a table space, consider using SMS for that table space. The DMS advantages with I/O and space management efficiency are not as important with small tables. The SMS advantages of allocating space one page at a time, and only when needed, are more attractive with smaller tables. If one of your tables is larger, or you need faster access to the data in the tables, then a DMS table space with a small extent size should be considered.

  You may wish to use a separate table space for each very large table and group all small tables together in a single table space. This separation also allows you to select an appropriate extent size based on the table space usage. (See "Choosing an Extent Size" on page 50 for additional information.)

- The type of data in the table.

  You may, for example, have tables containing historical data that is used infrequently and as a result the end-user may be willing to accept a longer response time for queries executed against this data. In this situation, you could use a different table space for the historical tables and assign this table space to less expensive physical devices that have slower access rates.

  Alternatively, you may be able to identify some essential tables which require high availability and fast response time. You may want to put these tables into a table space assigned to a fast physical device that can help support these important data requirements.

  Using DMS table spaces, you can also spread your table across three different table spaces: one for index data; one for LOB and long field data; one for regular

table data. This allows you to choose the table space characteristics and the physical devices supporting those table spaces to best suit the type of data. For example, you could put your index data on the fastest devices you have available, and as a result, obtain significant performance improvements. If you split a table across DMS table spaces, you should consider backing up and restoring all parts of the table together if ROLLFORWARD recovery is enabled. SMS table spaces do not support the spreading of your table across table spaces in this fashion.

- The administration requirements of your tables.

  Some administration functions can be performed at the table space level instead of the database or table level. For example, taking a back up of a table space instead of a database can help you make better use of your time and resources. It allows you to frequently back up table spaces with large volumes of changes, while only occasionally backing up tables spaces with very low volumes of changes.

  You may restore a database or a table space. If unrelated tables do not share table spaces, you have the ability to restore a smaller portion of your database, and as a result, reduce the time and resource requirements for the restore utility.

  A general rule-of-thumb could be to group related tables in a set of table spaces. These tables could be related through referential constraints, or through other business constraints defined on the tables using triggers.

  Another aspect to consider for administration of your tables, is how often you might want to drop and redefine a particular table. If the frequency is high, you may want to define the table in its own table space, since it is more efficient to drop a DMS table space than it is to drop a table.

Choosing an Extent Size

The extent size for a table space indicates the number of pages of table data that will be written to a container before data will be written to the next container. When selecting an extent size, you should consider:

- The size and type of tables in the table space.

  Space in DMS table spaces is allocated to a table an extent at a time. As the table is populated and an extent becomes full, a new extent is allocated.

  A table is made up of the following separate table objects

  – A DATA object. This is where the regular column data is stored.

  – An INDEX object. All indexes defined on the table are stored here.

  – A LONG FIELD object. If your table has one or more LONG columns, they are all stored here.

  – Two LOB objects. If your table has one or more LOB columns, they are stored in these two table objects:

    - One table object for the LOB data

    - A second table object for meta-data describing the LOB data

Each table object is stored separately, and therefore each allocates new extents as needed. Each table object is also paired up with a meta-data object called an *extent map*, which describes all the extents in the table space which belong to the table object. Space for extent maps is also allocated an extent at a time.

The initial allocation of space for a table, therefore, is two extents for each table object. If you have many small tables in a table space, you may have a relatively large amount of space allocated to store a relatively small amount of data. In such a case, you should specify a small extent size, or use an SMS table space which allocates pages one at a time.

If, on the other hand, you have a very large table that has a high growth rate, and you are using an DMS table space with a small extent size, you could have unnecessary overhead related to the frequent allocation of additional extents.

- The type of access to the tables.

  If access to the tables includes many queries or transactions that process large quantities of data, prefetching data from the tables may provide significant performance benefits. (See “Prefetching Data into the Buffer Pool” on page 405 for information about data prefetching and recommendations on its relationship to the extent size.)

- The minimum number of extents required.

  There must be enough space in the containers for five extents of the table space, otherwise the table space will not be created.

Recommendations for Catalog and Temporary Table Spaces

For each database, a single SMS temporary table space is recommended. SMS and not DMS, is recommended for the following reasons:

- Although DB2 supports multiple temporary table spaces, at runtime DB2 uses each temporary table space in turn and not at the same time. DB2 controls which temporary table space is used and not the user. Therefore each temporary table space must be large enough to accommodate the largest possible temporary table. As a result, it makes more sense to pool all the temporary space into one temporary table space. Allowing for multiple temporary table spaces is still useful when you want to change the definition of your table space. Since you must always have at least one table space, you must be able to have two table spaces to change the definition -- one with the old definition and one with the new definition.
- DB2 attempts to keep temporary tables in memory as much as possible. Since a DMS table space is comprised of pre-allocated storage space, and since you need to pre-allocate sufficient space to handle peak temporary space use, and since the pre-allocated space is not free for use for any other purpose, the choice of a DMS table space is not the best choice. With a SMS table space, temporary space is not pre-allocated but only consumed when needed. When not needed by the database manager, this space is free for other use.

DMS should only be considered if you need better performance than is possible if you use SMS.

For each database, a SMS table space for the catalogs is recommended. SMS and not DMS, is recommended for the following reasons:

- The database catalog consists of many tables of varying sizes. When using a DMS table space, a minimum of two extents are allocated for each table object. Depending on the extent size chosen, a significant amount of allocated and unused space may result. If using a DMS table space, then a small extent size (two to four pages) should be chosen; otherwise, a SMS table space should be used.
- There are large object (LOB) columns in the catalog tables. LOB data is not kept in the buffer pool with other data but is read from disk each time it is needed. Reading from disk slows down the performance of DB2 where the LOB columns of the catalogs are involved. Since a file system usually has its own place for storing (or caching) data, using a SMS table space, or a DMS table space built on file containers, make avoidance of I/O possible when the LOB has previously been referenced.

Given these considerations, a SMS table space is a slightly better choice for the catalogs.

Another factor to consider is if you will need to enlarge the catalog table space in the future. While some platforms have support for enlarging the underlying storage for SMS containers, and while the use of redirected restore to enlarge a SMS table space is available, the use of a DMS table space would allow for easier addition of new containers than the two other choices.

Workload Considerations

The primary type of workload being managed by DB2 in your environment can have an effect on your choice of the type of table space used. An online transaction process (OLTP) workload is characterized by transactions that make random access to data and that usually return small sets of data. Given that the access is random, and to one or a few pages, then prefetching is not possible. The important fact when considering I/O becomes the retrieving of a page of data with the minimum cost possible.

DMS table spaces using device containers perform best in this situation. DMS table spaces with file containers or SMS table spaces are also reasonable choices for OLTP workloads if maximum performance is not required. With little or no sequential I/O expected, the settings for the EXTENTSIZE and PREFETCHSIZE parameters on the CREATE TABLESPACE statement are not important for I/O efficiency.

A query workload is characterized by transactions that make sequential or partially sequential access to data and that usually return large sets of data. Efficient parallel prefetch should be possible in the type of table space chosen. A DMS table space using multiple device containers and where each container is on a separate disk, offers the greatest potential for efficient prefetching. The value of the PREFETCHSIZE parameter on the CREATE TABLESPACE statement should be set to the value of the EXTENTSIZE parameter multiplied by the number of device containers. This allows DB2 to prefetch from all containers in parallel.

A reasonable alternative with a query workload is to use files if the file system has its own prefetching. The files can be either of DMS type using file containers, or of SMS type. Note that if you use SMS, you need to have the directory containers map to separate physical disks in order to achieve I/O parallelism.

A mixed workload is characterized by transactions that are a mixture of the two types mentioned above. Your choice of SMS or DMS table spaces result from combining the considerations and advice from each of the two types of workload. Your goal will be to make single I/O requests as efficient as possible for OLTP workloads, and to maximize the efficiency of parallel I/O for the query workload.

Choosing an SMS or DMS Table Space

There are a number of trade-offs to consider when determining which type of table space you should use to store your data.

Advantages of a SMS Table Space:

- Space is not allocated by the system until it is required
- Creating a database requires less initial work since you do not have to predefine the containers.

Advantages of a DMS Table Space:

- The size of a table space can be increased by adding containers, using the ALTER TABLESPACE statement. Existing data is automatically rebalanced across the new set of containers to retain optimal I/O efficiency.
- A table can be split across multiple table spaces based on the type of data being stored:
  - Long field and LOB data
  - Indexes
  - Regular table data

  You might want to separate your table data for performance reasons, or to increase the amount of data stored for a table. For example, you could have a table with 64GB of regular table data, 64GB of index data and 2TB of long data.
- The location of the data on the disk can be controlled, if the operating system allows this.
- If all table data is in a single table space, a table space can be dropped and redefined with less overhead than dropping and redefining a table.
- In general, a well-tuned set of DMS table spaces will outperform SMS table spaces.

In general, small personal databases are easiest to manage with SMS table spaces. On the other hand, for large, growing databases you will probably only want to use SMS table spaces for the temporary table spaces and separate DMS table spaces, with multiple containers, for each table. In addition, long fields and indexes would be stored on their own table spaces.

If you choose to use DMS table spaces with device containers, you must be willing to tune and administer your environment. For more information, see “Performance Considerations for DMS Devices” on page 416.

Chapter 3. Implementing Your Design

After determining the design of your database, you must create the database and the objects within it. These objects include schemas, nodegroups, table spaces, tables, views, aliases, user-defined types (UDTs), user-defined functions (UDFs), triggers, constraints, indexes, and packages. You can create these objects using SQL statements in the command line processor, from the Control Center (on the Windows 95, Windows NT, and OS/2 operating systems), or through APIs in applications.

For information on SQL statements, see the *SQL Reference* manual. For information on command line processor commands and user APIs, see the *Command Reference* and *API Reference* manuals respectively.

Note: Your platform may support a user interface where you can create database objects. This interface can be used instead of the SQL statements, command line processor commands, or user APIs. Check the *Quick Beginnings* manual for your platform to determine if you have this capability.

The following topics are expanded and discussed in greater detail later in this chapter:

- Conceptual information you should know before you create a database
- How to Create Objects
- How to Alter Objects
- How to Delete Objects.

There may be operating system-specific differences with some of the topics discussed below in those areas where DB2 Universal Database interacts with the operating system. You may be able to take advantage of native operating system capabilities or differences beyond those offered by DB2 UDB. You should refer to your appropriate *Quick Beginnings* manuals and specific operating system documentation for precise differences.

As an example, Windows NT** supports an application type known as a “service.” DB2 for Windows NT can have a DB2 instance defined as a service. A service can be started automatically at system boot, by a user through the Services control panel applet, or by a Win32-based application that uses the service functions included in the Microsoft** Win32** application programming interface (API). Services can execute even when no user is logged on to the system.

Introductory Concepts for Database Implementation

Before you implement a database, you should understand the following concepts:

Starting and Stopping DB2

You may need to start or stop DB2 during normal business operations; for example, to do maintenance. To start DB2 on your system, enter the command:

```
db2start
```

This command can be run through the Control Center (on Windows 95, Windows NT, or OS/2 operating systems), or at the server as an operating system command or as a command line processor command. You must have SYSADM, SYSCTRL, or SYSMAINT authority to run this command.

To stop DB2 on your system, you must do the following:

1. Attach to an instance of the database. You do not require any special authorization for this.
2. Display all applications and users that are connected to the specific database that you want to stop. To ensure that no vital or critical applications are running, list applications. You need SYSADM, SYSCTRL and SYSMAINT authority for this.
3. Force all applications and users off the database. You require SYSADM and SYSCTRL authority to force users.
4. Stop the DB2 instance by typing the command:

   ```
   db2stop
   ```

The *db2stop* command can be run as an operating system command or as a Command Line Processor command. This command can only be run at the server. No database connections are allowed when running this command; however, if there are any instance attachments, they are forced off before DB2 is stopped.

Using Multiple Instances of the Database Manager

Multiple instances of the database manager may be created on a single server. This means that you can create several instances of the same product on a physical machine, and have them running concurrently. This provides flexibility in setting up environments.

You may wish to have multiple instances to:

- Separate your development environment from your production environment.
- Separately tune each for the specific applications it will service.
- Protect sensitive information from administrators. For example, you may wish to have your payroll database protected on its own instance so that owners of other instances will not be able to see payroll data.

DB2 program files are physically stored in one location on a particular machine. Each instance that is created points back to this location so the program files are not duplicated for each instance created. Several related databases can be located within a single instance.

Instances are cataloged as either local or remote in the node directory. Your default instance is defined by the DB2INSTANCE environment variable. You can *attach* to

other instances to perform maintenance and utility tasks that can only be done at an instance level, such as creating a database, forcing off applications, monitoring a database, or updating the database manager configuration. When you attempt to attach to an instance that is not in your default instance, the node directory is used to determine how to communicate with that instance.

To attach to another instance, which may be remote, use the ATTACH command as described in the *Command Reference* manual. For example,

```
attach to testdb2
```

will attach you to the instance called `testdb2` that was previously cataloged in the node directory.

After performing maintenance activities for the `testdb2` instance, you can then *detach* from that instance by executing the following command:

```
detach
```

The *Command Reference* provides information about the type of connection that is required to execute each command.

DB2 support for multiple instances varies by operating system. See the *Quick Beginnings* guide appropriate to your platform for information on defining multiple DB2 instances on one machine.

Organizing and Grouping Objects by Schema

The objects in a relational database are organized into *schemas*, which provide a logical classification of objects in the database. The schema is an object identified in the high-order part of a two-part object name. When an object such as a table, view, alias, distinct type, function, index, package or trigger is created, it is assigned to a schema. This assignment is done either explicitly or implicitly.

For example, USER A issues a CREATE TABLE statement in schema C as follows:

```
CREATE TABLE C.X (COL1 INT)
```

As described in “Definition of System Catalog Tables” on page 73, some objects are created within certain schemas when the database is created.

Before creating your own objects, you need to consider whether you want to create them in your default schema (identified by your user ID) or by using a separate schema that logically groups the objects. If you are creating objects that will be shared, using a different schema name can be very beneficial. For more information on how to explicitly create a schema, see “Creating a Schema” on page 79.

Enabling Intra-Partition Parallelism

You must modify configuration parameters to take advantage of parallelism within a database partition or within a non-partitioned database. For example, intra-partition parallelism can be used to take advantage of the multiple processors on a symmetric muti-processor (SMP) machine.

Use the GET DATABASE CONFIGURATION and the GET DATABASE MANAGER CONFIGURATION commands to find out the values of individual entries in a specific database, or in the database manager configuration file. To modify individual entries for a specific database or in the database manager configuration file, use the UPDATE DATABASE CONFIGURATION and the UPDATE DATABASE MANAGER CONFIGURATION commands respectively.

Configuration parameters that affect intra-partition parallelism include the *max_querydegree* and *intra_parallel* database manager parameters, and the *dft_degree* database parameter. For more information on configuration parameters, see Chapter 19, “Configuring DB2” on page 459.

Enabling Data Partitioning

When running in a multiple partition environment, you can create a database from any node that exists in the *db2nodes.cfg* file using the CREATE DATABASE command or the sqlecrea() application programming interface (API). For information, see the *Command Reference* and *API Reference* manuals.

Before creating a partitioned database, you must determine if you will be a local or remote client to the instance where the database is to be created. Second, you must attach to the instance. You must also select which database partition will be the catalog node for the database. The database partition to which you attach and execute the CREATE DATABASE command becomes the *catalog node* for that particular database.

The catalog node is the database partition on which all system catalog tables are stored. All access to system tables must go through this database partition.

If possible, you should create each database in a separate instance. If this is not possible (that is, you must create more than one database per instance), you should spread the catalog nodes among the available database partitions. Doing this reduces contention for catalog information at a single database partition.

Note: You should regularly do a backup of the catalog node and avoid putting data on it (whenever possible), because other data increases the time required for the backup.

When you create a database, it is automatically created across all the database partitions defined in the db2nodes.cfg file.

When the first database in the system is created, a system database directory is formed. It is appended with information about any other databases that you create. The system database directory is `sqldbdir` and is located in the `sqllib` directory under your home directory. This directory must reside on a shared file system, (for example, NFS on UNIX platforms) because there is only one system database directory for all the database partitions that make up the parallel database.

Also resident in the `sqldbdir` directory is the system intention file. It is called `sqldbins`, and ensures that the database partitions remain synchronized. The file must also reside

on a shared file system since there is only one directory across all database partitions. The file is shared by all the partitions making up the database.

Configuration parameters have to be modified to take advantage of data partitioning. Use the GET DATABASE CONFIGURATION and the GET DATABASE MANAGER CONFIGURATION commands to find out the values of individual entries in a specific database, or in the database manager configuration file. To modify individual entries in a specific database, or in the database manager configuration file, use the UPDATE DATABASE CONFIGURATION and the UPDATE DATABASE MANAGER CONFIGURATION commands respectively.

The database manager configuration parameters affecting a partitioned database include *conn_elapse*, *fcm_num_anchors*, *fcm_num_buffers*, *fcm_num_connect*, *fcm_num_rqb*, *max_connretries*, *max_coordagents*, *max_time_diff*, *num_poolagents*, and *stop_start_time*.

For more information on configuration parameters, see Chapter 19, “Configuring DB2” on page 459.

Before Creating a Database

Before creating a database, you should consider or carry out the following tasks:

- Design Logical and Physical Database Characteristics
- Create an Instance
- Establish Environment Variables and the Profile Registry
- DB2 Administration Server (DAS)
- Create a Node Configuration File
- Create a Database Configuration File
- Enable FCM Communications

Design Logical and Physical Database Characteristics

You must make logical and physical database design decisions before you create a database. To find out more about logical database design, see Chapter 1, “Designing Your Logical Database” on page 3. To find out more about physical database design, see Chapter 2, “Designing Your Physical Database” on page 25.

Create an Instance

As part of your installation procedure, you create an instance of DB2. It is possible to have more than one instance on a system. You may only work within one instance of DB2 at a time.

Use the **db2icrt** command to create an instance of DB2. When using this command, you should provide the login name of the instance owner and optionally specify the authentication type of the instance. The authentication type applies to all databases created under that instance. The authentication type is a statement of where the authenticating of users will take place. For more information on authentication, see Chapter 4, “ Controlling Database Access” on page 111. For more information on the `db2icrt` command, see the *Command Reference* manual.

Establish Environment Variables and the Profile Registry

Environment variables and registry values control your database environment.

Prior to the introduction of the DB2 profile registry, changing your environment variables on Windows or OS/2 workstations (for example) required you to change an environment variable and reboot. Now, your environment is controlled with a few exceptions by registry values stored in the DB2 profile registries. Use the **db2set** command to update registry values without rebooting; this information is stored immediately in the profile registries.

Note: The DB2 environment variables `db2instance`, `db2path`, and `db2instprof` may not, depending on the operating system, be stored in the DB2 profile registries. In order to update these environment variables, the **set** command must be used and the system rebooted.

Using the profile registry allows for centralized control of the environment variables. Appendix E, "DB2 Registry Values and Environment Variables" on page 633 lists many of the environment variables and registry values. Different levels of support are now provided through the different environment profiles. Remote administration of the environment variables is also available when using the DB2 Administration Server.

There are four (4) profile registries. They are:

- The DB2 Instance Level Profile Registry. The majority of the DB2 environment variables are placed within this registry. The environment variable settings for a particular instance are kept in this registry.
- The DB2 Global Level Profile Registry. If an environment variable is not set for a particular instance, this registry is used. This registry has the machine-wide environment variable settings.
- The DB2 Instance Node Level Profile Registry. In a system where the database is divided across different database partitions, this registry resides on every node (that is, machine), and contains environment variable settings for all instances storing data on the node.
- The DB2 Instance Names Registry. This registry contains a list of all instance names recognized by this system.

Users can override DB2 Instance Profile Registry environment variable settings for their session by changing session environment variable settings using the **db2set** command.

DB2 configures the operating environment by checking for registry values and environment variables and resolving them in the following order:

1. Environment variables set with the `set` command.
2. Registry values set with the instance node level profile (using the `db2set -I` command with a node number as shown below).
3. Registry values set with the `db2set` command.
4. Registry values set with the instance profile (using the `db2set -I` command as shown below).
5. Registry values set with the group profile (using the `db2set -G` command as shown below).

Using the db2set Command

The **db2set** command supports the local declaration of the environment variables to a particular setting.

To display help information for the command, use:

```
db2set ?
```

To list the complete set of all supported registry variables for your platform, use:

```
db2set -lr
```

To list all currently defined registry variables for this session, use:

```
db2set
```

To show the current session value of a registry variable, use:

```
db2set variable_name
```

To delete the current session value of a registry variable, use:

```
db2set variable_name=
```

To change a registry variable for this session only, use:

```
db2set variable_name=new_value
```

To change a registry variable default for all databases in the instance, use:

```
db2set variable_name=new_value
   -I instance_name
```

To change a registry variable default for all instances in the system, use:

```
db2set variable_name=new_value -G
```

Note: The two parameters "-I" and "-G" cannot be used at the same time in the same command.

To change a registry variable default for a particular node in an instance, use:

```
db2set variable_name=new_value
   -I instance_name node_number
```

To reset all registry variables for an instance back to the defaults found in the Global Profile Registry, use:

```
db2set -r variable_name
```

To reset all registry variables for a node in an instance back to the defaults found in the Global Profile Registry, use:

```
db2set -r variable_name node_number
```

Setting Environment Variables on OS/2

On OS/2, you should have no environment variables defined in config.sys apart from DB2PATH and DB2INSTPROF. All values should be defined in the profile registries using the **db2set** command except for those that remain true environment variables.

DB2INSTANCE also remains a true environment variable, however, it is not required if you make use of the DB2INSTDEF registry variable. This variable defines the default instance name to use if DB2INSTANCE is not set.

To set system environment variables, do the following: Edit the config.sys file, and reboot the system to have the change take effect.

The different profile registries are located according to the following:

- The DB2 Instance Level Profile Registry file is located under:

  `%DB2INSTPROF%\`*`instance_name`*`\PROFILE.ENV`

  Note: The *instance_name* is specific to the database partition you are working with.

- The DB2 Global Level Profile Registry is located under:

  `%DB2INSTPROF%\PROFILES.REG`

- The DB2 Instance Node Level Profile Registry is located under:

  `%DB2INSTPROF%\`*`instance_name`*`\NODES\`*`node_number`*`.ENV`

  Note: The *instance_name* and the *node_number* are specific to the database partition you are working with.

  There is an additional registry file that keeps track of all defined nodes. The information in this file is roughly equivalent to what is kept in the db2nodes.cfg file.

  `%DB2INSTPROF%\`*`instance_name`*`\NODES.CFG`

- The DB2 System Profile Registry is located under:

  `%DB2INSTPROF%\PROFILES.REG`

Remote registry support is allowed: Only the global level profile registry key, GLOBAL_PROFILE, must be local to a machine. The system profile registry key, PROFILES, may reside on a cluster or a remote machine registry. To specify the remote registry, set the DB2 system variable, DB2REMOTEPREG, to the remote machine name. For example,

```
db2set DB2REMOTEPREG=\\rmtwkstn
```

where *rmtwkstn* is the remote workstation name.

Note: Care should be taken in setting this option since all the DB2 instance profiles and instance listings will be located on the specified remote machine name.

This feature may be used in combination with setting DBINSTPROF to point to a remote LAN drive on the same machine that contains the registry.

Setting Environment Variables on Windows NT and Windows 95

On the Windows NT and Windows 95 operating systems, all DB2 environment values should be defined in the profile registries using the **db2set** command, except for those that are true environment variables. For Windows NT, you should not have the DB2 environment variables defined in either your machine's user or system environment variables sections. On Windows 95, you should not have DB2 environment variables defined in your autoexec.bat file.

To determine the settings of an environment variable, use the **echo** command. For example, to check the value of the `db2path` environment variable, enter:

```
echo %db2path%
```

To set system environment variables, do the following:

On Windows 95: Edit the autoexec.bat file, and reboot the system to have the change take effect.

On Windows NT 4.x: You can set the DB2 environment variables `db2instance`, `db2path`, and `db2instprof` as follows:

- Select **Start**, **Settings**, **Control Panel**.
- Double-click on the **System** icon.
- In the `System Control Panel`, in the `System Environment Variables` section, do the following:
  1. If the db2instance variable does not exist:
     a. Select any system environment variable.
     b. Change the name in the `Variable` field to db2instance.
     c. Change the `Value` field to the instance name, for example db2inst.
  2. If the db2instance variable already exists, append a new value:
     a. Select the db2instance environment variable.
     b. Change the `Value` field to the instance name, for example db2inst.
  3. Select Set.
  4. Select OK.
  5. Reboot your system for these changes to take effect.

The profile registries are located as follows:

- The DB2 Instance Level Profile Registry in the Windows NT operating system registry, with the path:

  `\HKEY_LOCAL_MACHINE\SOFTWARE\IBM\DB2\PROFILES\`*`instance_name`*

  Note: The *instance_name* is specific to the database partition you are working with.

- The DB2 Global Level Profile Registry in the Windows NT registry, with the path:

  `\HKEY_LOCAL_MACHINE\SOFTWARE\IBM\DB2\GLOBAL_PROFILE`

- The DB2 Instance Node Level Profile Registry in the Windows NT registry, with the path:

  ...\SOFTWARE\IBM\DB2\PROFILES*instance_name*\NODES*node_number*

  Note: The *instance_name* and the *node_number* are specific to the database partition you are working with.

- The DB2 System Profile Registry in the Windows NT operating system registry, with the path:

  \HKEY_LOCAL_MACHINE\SOFTWARE\IBM\DB2\PROFILES

Remote registry support is allowed: Only the global level profile registry key, GLOBAL_PROFILE, must be local to a machine. The system profile registry key, PROFILES, may reside on a cluster or a remote machine registry. To specify the remote registry, set the DB2 system variable, DB2REMOTEPREG, to the remote machine name. For example,

```
db2set DB2REMOTEPREG=\\rmtwkstn
```

where *rmtwkstn* is the remote workstation name.

Note: Care should be taken in setting this option since all DB2 instance profiles and instance listings will be located on the specified remote machine name.

This feature may be used in combination with setting DBINSTPROF to point to a remote LAN drive on the same machine that contains the registry.

Setting Environment Variables on UNIX Systems

The scripts **db2profile** (for Korn shell) and **db2cshrc** (for Bourne shell or C shell) are provided as examples to help you set up the database environment. You can find these files in `insthome/sqllib`, where `insthome` is the home directory of the instance owner.

These scripts include statements to:

- Update a user's path with the following directories:
  - `insthome/sqllib/bin`
  - `insthome/sqllib/adm`
  - `insthome/sqllib/misc`
- Set `db2instance` to the default local `instance_name` for execution.

An instance owner or SYSADM user may customize these scripts for all users of an instance. Alternatively, users can copy and customize a script, then invoke a script directly or add it to their .profile or .login files.

To change the environment variable for the current session, issue commands similar to the following:

- For Korn shell:

```
db2instance=inst1
export db2instance
```

- For Bourne shell or C shell:

  ```
  set db2instance inst1
  ```

In order for the DB2 profile registry to be administered properly, the following file ownership rules must be followed on UNIX operating systems. (For information on DB2 Administration Server (DAS), see "DB2 Administration Server (DAS).")

- The DB2 Instance Level Profile Registry file is located under:

  ```
  $INSTHOME/sqllib/profile.env
  ```

  The access permissions and ownership of this file should be:

  ```
  -rw-r--r-- Instance_Owner DAS_Instance_Group profile.env
  ```

  The *$INSTHOME* is the home path of the instance owner.

- The DB2 Global Level Profile Registry is located under:
  - */var/db2/v5/default.env* for AIX, Solaris, SINIX, and SCO operating systems.
  - */var/opt/db2/v5/default.env* for the HP-UX operating system.

  The access permissions and ownership of this file should be:

  ```
  -rw-r--r-- DAS_Instance_Owner DAS_Instance_Group default.env
  ```

- The DB2 Instance Node Level Profile Registry is located under:

  ```
  $INSTHOME/sqllib/nodes/node_number.env
  ```

  The access permissions and ownership of the directory and this file should be:

  ```
  drwxrwxr-x Instance_Owner DAS_Instance_Group nodes

  -rw-r--r-- Instance_Owner DAS_Instance_Group node_number.env
  ```

  Note: The *Instance_Owner* and the *DAS_Instance_Owner* should both be members of the *DAS_Instance_Group*.

  The *$INSTHOME* is the home path of the instance owner.

- The DB2 System Profile Registry is located under:
  - `/var/db2/v5/profiles.reg` for AIX, Solaris, SINIX, and SCO operating systems.
  - `/var/opt/db2/v5/profiles.reg` for the HP-UX operating system.

  The access permissions and ownership of this file should be:

  ```
  -rw-r--r-- root system profiles.reg
  ```

DB2 Administration Server (DAS)

DB2 Administration Server (DAS) is a DB2 instance that enables remote administration of DB2 servers. The Administration Server instance is created and used in a similar fashion to any other DB2 instance. You can only have one DAS on a machine.

For more information on setting up DAS communications, see the *Quick Beginnings* for your platform.

Creating the DAS

- On the OS/2 or Windows NT platforms:

  Enter `db2admin create`. (If this command returns an error stating that a DAS already exists, issue a "db2admin drop", then re-issue "db2admin create".)

  When creating the DAS, you can optionally provide a user account name and a user password. If valid, the user account name and password will identify the owner of the DAS. After you create the DAS, you can establish or modify its ownership by providing a user account name and user password with the **db2admin setid** command.

- On UNIX platforms:

  1. Ensure that you have root authority.
  2. At a command prompt, issue the following command from the `instance` subdirectory in the path of the DB2 Universal Database instance:

     ```
     dasicrt ASName
     ```

     where *ASName* is the instance name of the Administration Server.

Once you create an Administration Server, you should use it to establish directory structures and access permissions.

Starting and Stopping the DAS

To start the DAS, enter `db2admin start`

To stop the DAS, enter `db2admin stop`

Note: For both cases under UNIX, the person using these commands must have logged on with the authorization ID of the DAS owner.

Configuring the DAS

To see the current values for those database manager configuration parameters relevant to the DAS, enter:

```
db2 get admin cfg
```

To update individual entries in the database manager configuration file relevant to the DAS, enter:

```
db2 update admin cfg using ...
```

See the *Command Reference* for more information on which database manager configuration parameters can be modified.

To reset the configuration parameters to the recommended database manager defaults, enter:

```
db2reset admin cfg
```

Changes to the database manager configuration file become effective only after they are loaded into memory (that is, when `db2start` is executed).

To set up the communications protocols for the DAS, see the *Quick Beginnings* for your platform.

Cataloging the DB2 Administration Server

The DB2 Administration Server must be cataloged before the user can attach to the DAS. The only difference from the normal CATALOG NODE command is the addition of the keyword ADMIN after the keyword CATALOG. For more information on using the CATALOG command for nodes, including the differences for the various communication protocols, see the *Command Reference*.

When cataloging locally, you must know the instance name of the DAS. Use the db2set command to view the instance name:

```
db2set db2adminserver
```

Security Considerations for the DAS

Use the following command to associate a user ID with the DAS:

```
db2admin setid userid password
```

Note: Do **not** use the Windows NT operating system to set the user ID for the DAS. There is no guarantee that the user will receive all required privileges.

It is recommended that the user ID has SYSADM authority on each of the servers within the environment so that it can start or stop other instances if required.

Removing the DAS

To remove the DAS:

- On the OS/2 or Windows NT operating systems:
  1. Stop the DAS, using `db2admin stop`.
  2. Drop the DAS, using `db2admin drop`.
- On UNIX platforms:
  1. Ensure that you have root authority.
  2. From the `instance` subdirectory in the path of the DB2 Universal Database instance, issue:

     ```
     dasidrop ASName
     ```

     where the `ASName` is the instance name of the Administration Server.

Create a Node Configuration File

If your database is to operate in a partitioned database environment, you must create a node configuration file called `db2nodes.cfg`. This file must be located in the `sqllib` subdirectory of the home directory for the instance before you can start the database manager with parallel capabilities across multiple partitions. The file contains configuration information for all database partitions in an instance, and is shared by all database partitions for that instance.

Note: You should not create files or directories under the `sqllib` subdirectory other than those created by DB2 to prevent the loss of data if an instance is deleted. There are two exceptions. If your system supports stored procedures, put the stored procedure applications in the `function` subdirectory under the `sqllib` subdirectory. (For information on stored procedures, see "Stored Procedures" on page 296.) The other exception is when user-defined distinct functions (UDFs) have been created. UDF executables are allowed in the same directory.

The file contains one line for each database partition that belongs to an instance. Each line has the following format:

```
nodenum hostname [logical-port [netname]]
```

Tokens are delimited by blanks. The variables are:

`nodenum` The node number, which can be from 0 to 999, uniquely defines a node. Node numbers must be in ascending sequence. You can have gaps in the sequence.

Once a node number is assigned, it cannot be changed. (Otherwise the information in the partitioning map, which specifies how data is partitioned, would be compromised.)

If you drop a node, its node number can be used again for any new node that you add.

The node number is used to generate a node name in the database directory. It has the format:

```
NODEnnnn
```

The *nnnn* is the node number, which is left-padded with zeros. This node number is also used by the CREATE DATABASE and DROP DATABASE commands.

`hostname` The hostname of the IP address for inter-partition communications. (There is an exception when `netname` is specified. In this situation, `netname` is used for most communications, with `hostname` only being used for DB2START, DB2STOP, and `db2_all`.)

`logical-port` This parameter is optional, and specifies the logical port number for the node. This number is used with the database manager instance name to identify a TCP/IP service name entry in the `etc/services` file.

The combination of the IP address and the logical port is used as a well-known address, and must be unique among all applications to support communications connections between nodes.

For each *hostname*, one *logical-port* must be either 0 (zero) or blank (which defaults to 0). The node associated with this *logical-port* is the default node on the host to which clients connect. You can override this with the DB2NODE environment variable in `db2profile` script, or with the sqlesetc() API.

If you have multiple nodes on the same host (that is, more than one *nodenum* for a host), you should assign the *logical-port* numbers to the logical nodes in ascending order, from 0, with no gaps.

`netname` This parameter is optional, and is used to support a host that has more than one active TCP/IP interface, each with its own hostname.

The following example shows a possible node configuration file for an RS/6000 SP system on which SP2EN1 has multiple TCP/IP interfaces, two logical nodes, and uses SP2SW1 as the DB2 Universal Database interface. It also shows the node numbers starting at 1 (rather than at 0), and a gap in the *nodenum* sequence:

```
nodenum   hostname   logical-port   netname
1         SP2EN1     0              SP2SW1
2         SP2EN1     1              SP2SW1
4         SP2EN2     0
5         SP2EN3
```

You can update the `db2nodes.cfg` file using an editor of your choice. You must be careful, however, to protect the integrity of the information in the file, as data partitioning requires that the node number not be changed. The node configuration file is locked when you issue DB2START and unlocked after DB2STOP ends the database manager. The DB2START command can update the file, if necessary, when the file is locked. For example, you can issue DB2START with the RESTART option or the ADDNODE option.

Note: If the DB2STOP command is not successful and does not unlock the node configuration file, issue DB2STOP FORCE to unlock it.

Create a Database Configuration File

A *database configuration file* is also created for each database. This file contains values for various *configuration parameters* that affect the use of the database, such as:

- Parameters specified and/or used when creating the database (for example, database code page, collating sequence, DB2 release level)
- Parameters indicating the current state of the database (for example, backup pending flag, database consistency flag, roll-forward pending flag)
- Parameters defining the amount of system resources that the operation of the database may use (for example, buffer pool size, database logging, sort memory size).

These parameters are described in detail in Chapter 19, “Configuring DB2” on page 459, and throughout this book.

Performance Tip: Many of the configuration parameters come with default values, but may need to be updated to achieve optimal performance for your database.

On the Windows NT and OS/2 platforms, use the Performance Configuration SmartGuide, which helps you tune performance and balance memory requirements for

a single database per instance by suggesting which configuration parameters to modify and providing suggested values for them. To use this SmartGuide:

1. From the Control Center, click with mouse button 2 on the database for which you want to configure performance.
2. Select **Configure Performance** from the pop-up menu. The Performance Configuration SmartGuide opens.
3. Follow the steps in the SmartGuide and answer the questions it asks. (See the *Administration Getting Started* for a list of these questions.)
4. Note that if you select to update the parameters, they are not updated until:
   - For database parameters, the first new connection to the database after all applications were disconnected.
   - For database manager parameters, the next time you stop and start the instance.

In most cases the values recommended by the Performance Configuration SmartGuide will provide better performance than the default values, because they are based on information about your workload and you own particular server. However, note that the values are designed to improve the performance of, though not necessarily optimize, your database system. They should be thought of as a starting point on which you can make further adjustments to obtain optimized performance.

For details on how to refine your system by benchmarking, and to configure your system, see Chapter 18, “Benchmark Testing” on page 447 and Chapter 19, “Configuring DB2” on page 459.

For multiple partitions: When you have a database that is partitioned across more than one partition, the configuration file should be the same on all database partitions. Consistency is required since the SQL compiler compiles distributed SQL statements based on information in the local node configuration file and creates an access plan to satisfy the needs of the SQL statement. Maintaining different configuration files on database partitions could lead to different access plans, depending on which database partition the statement is prepared. Use **db2_all** to create the same configuration file on all database partitions.

Enable FCM Communications

In a partitioned database environment, most communication between database partitions is handled by the Fast Communications Manager (FCM). To enable the FCM at a database partition and allow communication with other database partitions, you must create a service directory in the partition's `/etc/services` file as shown below. The FCM uses the specified port to communicate. If you have defined multiple partitions on the same host, you must define a range of ports as shown below. The syntax of a service entry is as follows:

```
DB2_instance port/tcp #comment
```

DB2_*instance*

The value for *instance* is the name of the database manager instance. All characters in the name must be lowercase. Assuming an instance name of `db2puser`, you would specify `DB2_db2puser`

port/**tcp** The TCP/IP port that you want to reserve for the database partition.

#*comment* Any comment that you want to associate with the entry. The comment must be preceded by a pound sign (#).

If the `/etc/services` file is shared, you must ensure that the number of ports allocated in the file is either greater than or equal to the largest number of multiple database partitions in the instance. When allocating ports, also ensure that you account for any processor that can be used as a backup.

If the `/etc/services` file is not shared, the same considerations apply, with one additional consideration: you must ensure that the entries defined for the DB2 instance are the same in all `/etc/services` files (though other entries that do not apply to your partitioned database do not have to be the same).

If you have multiple database partitions on the same host in an instance, you must define more than one port for the FCM to use. To do this, include two lines in the `etc/services` file to indicate the range of ports you are allocating. The first line specifies the first port, while the second line indicates the end of the block of ports. In the following example, five ports are allocated for the instance `sales`. This means no processor in the instance has more than five database partitions.

```
DB2_sales          9000/tcp
DB2_sales_END      9004/tcp
```

Note: You must specify `END` in uppercase only. Also you must ensure that you include both underscore (_) characters.

Creating a Database

Creating a database sets up all the system catalog tables that are needed by the database and allocates the database recovery log. The database configuration file is created, and the default values are set. The database manager will also bind the database utilities to the database.

The following database privileges are automatically granted to PUBLIC: CREATETAB, BINDADD, CONNECT, and IMPLICIT_SCHEMA. SELECT privilege on the system catalog views is also granted to PUBLIC.

The following command line processor command creates a database called `personl`, in the default location, with the associated comment "Personnel DB for BSchiefer Co".

```
create database personl
   with "Personnel DB for BSchiefer Co"
```

The tasks carried out by the database manager when you create a database are discussed in the following sections:

- "Definition of Initial Nodegroups" on page 72
- "Definition of Initial Table Spaces"
- "Definition of System Catalog Tables" on page 73
- "Local Database Directory" on page 74
- "System Database Directory" on page 74
- "Definition of Database Recovery Log" on page 75
- "Binding Utilities to the Database" on page 75
- "Creating Nodegroups" on page 77

For additional information related to the physical implementation of your database, see Chapter 2, "Designing Your Physical Database" on page 25.

If you wish to create a database in a different, possibly remote, database manager instance, see "Using Multiple Instances of the Database Manager" on page 56. This topic also provides an introduction to the command you need to use if you want to perform any instance-level administration against an instance other than your default instance, including remote instances.

Note: See the *Command Reference* for information about the default database location and about specifying a different location with the CREATE DATABASE command.

Definition of Initial Nodegroups

When a database is initially created, database partitions are created for all partitions specified in the db2nodes.cfg file. Other partitions can be added or removed with the ADD NODE and DROP NODE commands.

Three nodegroups are defined:

- IBMCATGROUP for the SYSCATSPACE table space, holding system catalog tables
- IBMTEMPGROUP for the TEMPSPACE1 table space, holding temporary tables created during database processing
- IBMDEFAULTGROUP for the USERSPACE1 table space, by default holding user tables and indexes.

Definition of Initial Table Spaces

When a database is initially created, three table spaces are defined:

- SYSCATSPACE for the system catalog tables (see "Definition of System Catalog Tables" on page 73)
- TEMPSPACE1 for temporary tables created during database processing.
- USERSPACE1 for user-defined tables and indexes

If you do not specify any table space parameters with the CREATE DATABASE command, the database manager will create these table spaces using system managed storage (SMS) directory containers. These directory containers will be created in the subdirectory created for the database (see "Database Physical Directories" on page 25). The extent size for these table spaces will be set to the default.

If you do not want to use the default definition for these table spaces, you may specify their characteristics on the CREATE DATABASE command. For example, the following command could be used to create your database on OS/2:

```
CREATE DATABASE PERSONL
  CATALOG TABLESPACE
    MANAGED BY SYSTEM USING ('d:\pcatalog','e:\pcatalog')
    EXTENTSIZE 16 PREFETCHSIZE 32
  USER TABLESPACE
    MANAGED BY DATABASE USING (FILE'd:\db2data\personl' 5000,
                               FILE'd:\db2data\personl' 5000)
    EXTENTSIZE 32 PREFETCHSIZE 64
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING ('f:\db2temp\personl')
  WITH "Personnel DB for BSchiefer Co"
```

In this example, the definition for each of the initial table spaces is explicitly provided. You only need to specify the table space definitions for those table spaces for which you do not want to use the default definition.

The coding of the MANAGED BY phrase on the CREATE DATABASE command follows the same format as the MANAGED BY phrase on the CREATE TABLESPACE command. For additional examples, see “Creating a Table Space” on page 77.

Before creating your database, see “Designing and Choosing Table Spaces” on page 38.

Definition of System Catalog Tables

A set of system catalog tables is created and maintained for each database. These tables contain information about the definitions of the database objects (for example, tables, views, indexes, and packages), and security information about the type of access users have to these objects. These tables are stored in the SYSCATSPACE table space,

These tables are updated during the operation of a database; for example, when a table is created. You cannot explicitly create or drop these tables, but you can query and view their content. When the database is created, in addition to the system catalog table objects, the following database objects are defined in the system catalog:

- A set of user-defined functions (UDFs) is created in the SYSFUN schema. For more information about these system-created functions, see the *SQL Reference* manual.
- A set of read-only views for the system catalog tables is created in the SYSCAT schema. See Appendix I, “Catalog Views” on page 697 for information about these views.
- A set of updatable catalog views is created in the SYSSTAT schema. These updatable views allow you to update certain statistical information to investigate the performance of a hypothetical database, or to update statistics without using the RUNSTATS utility. See “Updatable Catalog Views” on page 698.

After your database has been created, you may wish to limit the access to the system catalog views, as described in “Securing the System Catalog Views” on page 138.

Definition of Database Directories

Three directories are used when establishing or setting up a new database.

- Local Database Directory
- System Database Directory
- Node Directory

Local Database Directory

A *local database directory* file exists in each path (or drive on other platforms) in which a database has been defined. This directory contains one entry for each database accessible from that location. Each entry contains:

- The database name provided with the CREATE DATABASE command
- The database alias name (which is the same as the database name, if an alias name is not specified)
- A comment describing the database, as provided with the CREATE DATABASE command
- The name of the root directory for the database
- Other system information.

To see the contents of this file for a particular database, issue the following command, where *location* specifies the location of the database:

```
LIST DATABASE DIRECTORY ON location
```

System Database Directory

A *system database directory* file exists for each instance of the database manager, and contains one entry for each database that has been cataloged for this instance. Databases are implicitly cataloged when the CREATE DATABASE command is issued and can also be explicitly cataloged with the CATALOG DATABASE command. For information about cataloging databases, see “Cataloging a Database” on page 76.

For each database created, an entry is added to the directory containing the following information:

- The database name provided with the CREATE DATABASE command
- The database alias name (which is the same as the database name)
- The database comment provided with the CREATE DATABASE command
- The location of the *local database directory*
- An indicator that the database is *indirect*, which means that it resides on the same machine as the system database directory file
- Other system information.

To see the contents of this file, issue the LIST DATABASE DIRECTORY command **without** specifying the location of the database directory file.

In a partitioned database environment, you must ensure that all database partitions always access the same system database directory file, `sqldbdir`, in the `sqldbdir` subdirectory of the home directory for the instance. Unpredictable errors can occur if either the system database directory or the system intention file `sqldbins` in the same `sqldbdir` subdirectory are symbolic links to another file that is on a shared file system. These files are described in “Enabling Data Partitioning” on page 58.

Node Directory

The database manager creates the node directory when the first database partition is cataloged. To catalog a database partition, use the CATALOG NODE command. To list the contents of the local node directory, use the LIST NODE DIRECTORY command. The node directory is created and maintained on each database client. The directory contains an entry for each remote workstation having one or more databases that the client can access. The DB2 client uses the communication end point information in the node directory whenever a database connection or instance attachment is requested.

The entries in the directory also contain information on the type of communication protocol to be used to communicate from the client to the remote database partition. Cataloging a local database partition creates an alias for an instance that resides on the same machine. A local node should be cataloged when there is more than one instance on the same workstation to be accessed from the user's client.

Definition of Database Recovery Log

A *database recovery log* keeps a record of all changes made to a database, including the addition of new tables or updates to existing ones. This log is made up of a number of *log extents*, each contained in a separate file called a *log file*.

The database recovery log can be used to ensure that a failure (for example, a system power outage or application error) does not leave the database in an inconsistent state. In case of a failure, the changes already made but not committed are rolled back, and all committed transactions, which may not have been physically written to disk, are redone. These actions ensure the integrity of the database.

For more information, see Chapter 6, “ Recovering a Database” on page 179.

Binding Utilities to the Database

When a database is created, the database manager attempts to bind the utilities in `db2ubind.lst` to the database. This file is stored in the `bnd` subdirectory of your `sqllib` directory.

Binding a utility creates a *package*, which is an object that includes all the information needed to process specific SQL statements from a single source file.

Note: If you wish to use these utilities from a client, you must bind them explicitly. See the *Quick Beginnings* manual appropriate to your platform for information.

If for some reason you need to bind or rebind the utilities to a database, issue the following commands using the command line processor:

```
connect to sample
bind @db2ubind.lst
```

Note: You must be in the directory where these files reside to create the packages in the `sample` database. The bind files are found in the `BND` subdirectory of the `SQLLIB` directory. In this example, `sample` is the name of the database.

Cataloging a Database

When you create a new database, it is automatically cataloged in the system database directory file. You may also use the CATALOG DATABASE command to explicitly catalog a database in the system database directory file. The CATALOG DATABASE command allows you to catalog a database with a different alias name, or to catalog a database entry that was previously deleted using the UNCATALOG DATABASE command.

The following command line processor command catalogs the `personl` database as `humanres`:

```
catalog database personl as humanres
   with "Human Resources Database"
```

Here, the system database directory entry will have `humanres` as the database alias, which is different from the database name (`personl`).

You can also catalog a database on an instance other than the default. In the following example, connections to database `B` are to `INSTANCE_C`.

```
catalog database b as b at node instance_c
```

Note: The CATALOG DATABASE command is also used on client nodes to catalog databases that reside on database server machines. For more information, see the *Quick Beginnings* manual appropriate to your platform.

For information on the Distributed Computing Environment (DCE) cell directory, see “DCE Directory Services” on page 77 and Appendix F, “Using Distributed Computing Environment (DCE) Directory Services” on page 647.

Note: To improve performance, you may cache directory files, including the database directory, in memory. (See “Directory Cache Support (dir_cache)” on page 497 for information about enabling directory caching.) When directory caching is enabled, a change made to a directory (for example, using a CATALOG DATABASE or UNCATALOG DATABASE command) by another application may not become effective until your application is restarted. To refresh the directory cache used by a command line processor session, issue a `db2 terminate` command.

In addition to the application level cache, a database manager level cache is also used for internal, database manager look-up. To refresh this “shared” cache, issue the `db2stop` and `db2start` commands.

For more information about directory caching, see “Directory Cache Support (dir_cache)” on page 497.

DCE Directory Services

DCE is an Open Systems Foundation** (OSF**) architecture that provides tools and services to support the creation, use, and maintenance of applications in a distributed heterogeneous computing environment. It is a layer between the operating system, the network, and a distributed application that allows client applications to access remote servers.

With local directories, the physical location of the target database is individually stored on each client workstation in the database directory and node directory. The database administrator can therefore spend a large amount of time updating and changing these directories. The DCE directory services provide a central directory alternative to the local directories. It allows information about a database or a database manager instance to be recorded once in a central location, and any changes or updates to be made at that one location.

DCE is not a prerequisite for running DB2, but if you are operating in a DCE environment, see Appendix F, "Using Distributed Computing Environment (DCE) Directory Services" on page 647 for more information.

Creating Nodegroups

You create a nodegroup with the CREATE NODEGROUP statement. This statement specifies the set of nodes on which the table space containers and table data are to reside. This statement also:

- Creates a partitioning map for the nodegroup. For details about the partitioning map, see "Partitioning Maps" on page 34.
- Generates a partitioning map ID.
- Inserts records into the following catalog tables:
  - SYSCAT.NODEGROUPS
  - SYSCAT.PARTITIONMAPS
  - SYSCAT.NODEGROUPDEF

Assume that you want to load some tables on a subset of the database partitions in your database. You would use the following command to create a nodegroup of two nodes (1 and 2) in a database consisting of at least 3 (0 to 2) nodes:

```
CREATE NODEGROUP mixng12 ON NODES (1,2)
```

For more information about creating nodegroups, see the *SQL Reference* manual.

The CREATE DATABASE command or sqlecrea() API also create the default system nodegroups, IBMDEFAULTGROUP, IBMCATGROUP, and IBMTEMPGROUP. (See "Designing and Choosing Table Spaces" on page 38 for information.)

Creating a Table Space

Creating a table space within a database assigns containers to the table space and records its definitions and attributes in the database system catalog. You can then create tables within this table space.

The syntax of the CREATE TABLESPACE statement is discussed in detail in the *SQL Reference* manual. For information on SMS and DMS table spaces, see “Designing and Choosing Table Spaces” on page 38.

The following SQL statement creates an SMS table space on OS/2 or Windows NT using three directories on three separate drives:

```
CREATE TABLESPACE RESOURCE
   MANAGED BY SYSTEM
   USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
```

The following SQL statement creates a DMS table space on OS/2 using two file containers each with 5,000 pages:

```
CREATE TABLESPACE RESOURCE
   MANAGED BY DATABASE
   USING (FILE'd:\db2data\acc_tbsp' 5000,
          FILE'e:\db2data\acc_tbsp' 5000)
```

In the above two examples, explicit names have been provided for the containers. You may also specify relative container names, in which case, the container will be created in the subdirectory created for the database (see “Database Physical Directories” on page 25).

In addition, if part of the path name specified does not exist, the database manager will create it. If a subdirectory is created by the database manager, it may also be deleted by the database manager when the table space is dropped.

The assumption in the above examples is that the table spaces are not associated with a specific nodegroup. The default nodegroup IBMDEFAULTGROUP is used when the following parameter is not specified in the statement:

```
IN nodegroup
```

The following SQL statement creates a DMS table space on a UNIX-based system using three logical volumes of 10 000 pages each, and specifies their I/O characteristics:

```
CREATE TABLESPACE RESOURCE
   MANAGED BY DATABASE
   USING (DEVICE '/dev/rdblv6' 10000,
          DEVICE '/dev/rdblv7' 10000,
          DEVICE '/dev/rdblv8' 10000)
   OVERHEAD 24.1
   TRANSFERRATE 0.9
```

The UNIX devices mentioned in this SQL statement must already exist and be able to be written to by the instance owner and the SYSADM group.

The following example creates a DMS table space on a nodegroup called ODDNODEGROUP in a UNIX partitioned database. ODDNODEGROUP must be previously created with a CREATE NODEGROUP statement. In this case, the ODDNODEGROUP nodegroup is assumed to be made up of database partitions

numbered 1, 3, and 5. On all database partitions, use the device `/dev/hdisk0` for 10 000 4K pages. In addition, declare a device for each database partition of 40 000 4K pages.

```
CREATE TABLESPACE PLANS
   MANAGED BY DATABASE
   USING (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n1hd01' 40000) ON NODE 1
         (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n3hd03' 40000) ON NODE 3
         (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n5hd05' 40000) ON NODE 5
```

UNIX devices are classified into two categories: character serial devices and block-structured devices. For all file-system devices, it is normal to have a corresponding character serial device (or *raw* device) for each block device (or *cooked* device). The block-structured devices are typically designated by names similar to “hd0” or “fd0.” The character serial devices are typically designated by names similar to “rhd0,” “rfd0,” or “rmt0.” These character serial devices have faster access than block devices. The character serial device names should be used on the CREATE TABLESPACE command and not block device names.

The overhead and transfer rate help to determine the best access path to use when the SQL statement is compiled. For information on the OVERHEAD and TRANSFERRATE parameters, see Part 3, “Tuning Application Performance” on page 263.

DB2 can greatly improve the performance of sequential I/O using the sequential prefetch facility, which uses parallel I/O. See “Understanding Sequential Prefetching” on page 406 for details on this facility.

The ALTER TABLESPACE SQL statement can be used to add a container to a DMS table space and modify the PREFETCHSIZE, OVERHEAD, and TRANSFERRATE settings for a table space. The transaction issuing the table space statement should be committed as soon as possible, to prevent system catalog contention.

Note: The PREFETCHSIZE should be a multiple of the EXTENTSIZE. For example if the EXTENTSIZE is 10, the PREFETCHSIZE should be 20 or 30. For more information, see “Understanding Sequential Prefetching” on page 406.

Creating Table Spaces in Nodegroups

By placing a table space in a multiple database partition nodegroup, all of the tables within the table space are divided or partitioned across each database partition in the nodegroup. The table space is created into a nodegroup. Once in a nodegroup, the table space must remain there; It cannot be changed to another nodegroup. The CREATE TABLESPACE statement is used to associate a table space with a nodegroup.

Creating a Schema

While organizing your data into tables, it may also be beneficial to group tables (and other related objects) together. This is done by defining a schema through the use of the CREATE SCHEMA statement. Information about the schema is kept in the system catalog tables of the database to which you are connected. As other objects are created, they can be placed within this schema.

The syntax of the CREATE SCHEMA statement is described in detail in the *SQL Reference* manual. The new schema name cannot already exist in the system catalogs and it cannot begin with "SYS".

If a user has SYSADM or DBADM authority, then the user can create a schema with any valid name. When a database is created, IMPLICIT_SCHEMA authority is granted to PUBLIC (that is, to all users).

The definer of any objects created as part of the CREATE SCHEMA statement is the schema owner. This owner can GRANT and REVOKE schema privileges to other users.

The following is an example of a CREATE SCHEMA statement that creates a schema for an individual user with the authorization ID "joe":

```
CREATE SCHEMA joeschma AUTHORIZATION joe
```

This statement must be issued by a user with DBADM authority.

Schemas may also be implicitly created when a user has IMPLICIT_SCHEMA authority. With this authority, users implicitly create a schema whenever they create an object with a schema name that does not already exist.

If users do not have IMPLICIT_SCHEMA authority, they can create a schema using their own authorization ID.

Creating a Table

After you determine how to organize your data into tables, the next step is to create those tables, by using the CREATE TABLE statement. The table descriptions are stored in the system catalog of the database to which you are connected.

The syntax of the CREATE TABLE statement is described in detail in the *SQL Reference*. For information about naming tables, columns, and other database objects, see Appendix D, “Naming Rules” on page 629.

The CREATE TABLE statement gives the table a name, which is a qualified or unqualified identifier, and a definition for each of its columns. You can store each table in a separate table space, so that a table space will contain only one table. If a table will be dropped and created often, it is more efficient to store it in a separate table space and then drop the table space instead of the table. You can also store many tables within a single table space. In a partitioned database environment, the table space chosen also defines the nodegroup and the database partitions on which table data is stored.

The table does not contain any data at first. To add rows of data to it, use one of the following:

- The INSERT statement, described in the *SQL Reference*
- The LOAD or IMPORT commands, described in the *Command Reference*

It is possible to add data into the table without logging the change. This is done using the NOT LOGGED INITIALLY parameter on the CREATE TABLE statement. Any changes made to the table by an INSERT, DELETE, UPDATE, CREATE INDEX, DROP INDEX, or ALTER TABLE operation in the same unit of work in which the table is created are not logged. Logging begins in subsequent units of work.

A table consists of one or more column definitions. A maximum of 500 columns can be defined for a table. Columns represent the attributes of an entity. The values in any column are all the same type of information. See the *SQL Reference* for more information.

A column definition includes a *column name, data type*, and any necessary *null attribute*, or default value (optionally chosen by the user).

The column name describes the information contained in the column and should be something that will be easily recognizable. It must be unique within the table; however, the same name can be used in other tables. See “Object Names” on page 631 for information about naming rules.

The data type of a column indicates the length of the values in it and the kind of data that is valid for it. The database manager uses character string, numeric, date, time and large object data types. Graphic string data types are only available for database environments using multi-byte character sets. In addition, columns can be defined with user-defined distinct types, which are discussed in “Creating a User-Defined Type (UDT)” on page 92.

The default attribute specification indicates what value is to be used if no value is provided. The default value can be specified, or a system-defined default value used. Default values may be specified for columns with, and without, the null attribute specification.

The null attribute specification indicates whether or not a column can contain null values.

The following is an example of a CREATE TABLE statement that creates the EMPLOYEE table in the RESOURCE table space. This table is defined in the sample database:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10M)   NOT NULL)
IN RESOURCE
```

The following sections build on the previous example to cover other options you should consider:

Large Object (LOB) Column Considerations

Before creating a table that contains large object columns, you need to make the following decisions:

1. Do you want to log changes to LOB columns?

   If you do not want to log these changes, you must turn logging off by specifying the NOT LOGGED clause when you create the table. For example:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10M)   NOT NULL  NOT LOGGED)
IN RESOURCE
```

   If the LOB column is larger than 1 GB, logging must be turned off. (As a rule of thumb, you may not want to log LOB columns larger than 10 MB.) As with other options specified on a column definition, the only way to change the logging option is to re-create the table.

   Even if you choose not to log changes, LOB columns are *shadowed* to allow changes to be rolled back, whether the roll back is the result of a system generated error, or an application request. Shadowing is a recovery technique where current storage page contents are never overwritten. That is, old, unmodified pages are kept as “shadow” copies. These copies are discarded when they are no longer needed to support a transaction rollback.

   Note: When recovering a database using the RESTORE and ROLLFORWARD commands, LOB data that was “NOT LOGGED” and was written since the last backup will be **replaced by binary zeros**.

2. Do you want to minimize the space required for the LOB column?

   You can make the LOB column as small as possible using the COMPACT clause on the CREATE TABLE statement. For example:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10M)   NOT NULL  NOT LOGGED  COMPACT)
IN RESOURCE
```

There is a **performance cost** when appending to a table with a compact LOB column, particularly if the size of LOB values are increased (because of storage adjustments that must be made).

On platforms such as OS/2 where sparse file allocation is not supported and where LOBs are placed in SMS table spaces, consider using the COMPACT clause. Sparse file allocation has to do with how physical disk space is used by an operating system. An operating system that supports sparse file allocation does not use as much physical disk space to store LOBs as compared to an operating system not supporting sparse file allocation. The COMPACT option allows for even greater physical disk space “savings” regardless of the support of sparse file allocation. Because you can get some physical disk space savings when using COMPACT, you should consider using COMPACT if your operating system does not support sparse file allocation.

Note: DB2 system catalogs for Version 5 use LOB columns and may take up more space than in previous versions.

Defining Constraints

This section discusses how to define constraints:

- “Defining a Unique Constraint”
- “Defining Referential Constraints” on page 84
- “Defining a Table Check Constraint” on page 87

For more information on constraints, see “Planning for Constraint Enforcement” on page 16 and the *SQL Reference*.

Defining a Unique Constraint: *Unique constraints* ensure that every value in the specified key is unique. A table can have any number of unique constraints, with at most one unique constraint defined as a primary key.

You define a unique constraint with the UNIQUE clause in the CREATE TABLE or ALTER TABLE statements. The unique key can consist of more than one column. More than one unique constraint is allowed on a table.

Once established, the unique constraint is enforced automatically by the database manager when an INSERT or UPDATE statement modifies the data in the table. The unique constraint is enforced through a unique index.

When a unique constraint is defined in an ALTER TABLE statement and an index exists on the same set of columns of that unique key, that index becomes the unique index and is used by the constraint.

You can take any one unique constraint and use it as the *primary key*. The primary key can be used as the parent key in a referential constraint (along with other unique constraints). There can be only one primary key per table. You define a primary key with the PRIMARY KEY clause in the CREATE TABLE or ALTER TABLE statement. The primary key can consist of more than one column.

A primary index forces the value of the primary key to be unique. When a table is created with a primary key, the database manager creates a primary index on that key.

Some performance tips for indexes used as unique constraints include:

- If you are using the IMPORT command, or the INSERT mode with the LOAD command, for an initial large load of data, create the unique key after the data has been imported or loaded. This avoids the overhead of maintaining the index while the table is being loaded. It also results in the index using the least amount of storage.
- If you are using the LOAD utility in REPLACE mode, create the unique key before loading the data. In this case, creation of the index during the load is more efficient than using the CREATE INDEX statement after the load.
- When loading a small amount of data to an existing table, you can improve performance if you use the IMPORT utility instead of LOAD's INSERT mode.

Defining Referential Constraints: Referential integrity is imposed by adding referential constraints to table and column definitions. Referential constraints are established with the the FOREIGN KEY Clause, and the REFERENCES Clause in the CREATE TABLE or ALTER TABLE statements.

The identification of foreign keys enforces constraints on the values within the rows of a table or between the rows of two tables. The database manager checks the constraints specified in a table definition and maintains the relationships accordingly. The goal is to maintain integrity whenever one database object references another.

For example, primary and foreign keys each have a department number column. For the EMPLOYEE table, the column name is WORKDEPT, and for the DEPARTMENT table, the name is DEPTNO. The relationship between these two tables is defined by the following constraints:

- There is only one department number for each employee in the EMPLOYEE table, and that number exists in the DEPARTMENT table.
- Each row in the EMPLOYEE table is related to no more than one row in the DEPARTMENT table. There is a unique relationship between the tables.
- Each row in the EMPLOYEE table that has a non-null value for WORKDEPT is related to a row in the DEPTNO column of the DEPARTMENT table.
- The DEPARTMENT table is the parent table, and the EMPLOYEE table is the dependent table.

The SQL statement defining the parent table, DEPARTMENT, is:

```
CREATE TABLE DEPARTMENT
   (DEPTNO    CHAR(3)     NOT NULL,
    DEPTNAME  VARCHAR(29) NOT NULL,
    MGRNO     CHAR(6),
    ADMRDEPT  CHAR(3)     NOT NULL,
    LOCATION  CHAR(16),
       PRIMARY KEY (DEPTNO))
IN RESOURCE
```

The SQL statement defining the dependent table, EMPLOYEE, is:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10m)   NOT NULL,
       FOREIGN KEY DEPT (WORKDEPT)
       REFERENCES DEPARTMENT ON DELETE NO ACTION)
IN RESOURCE
```

By specifying the DEPTNO column as the primary key of the DEPARTMENT table and WORKDEPT as the foreign key of the EMPLOYEE table, you are defining a referential constraint on the WORKDEPT values. This constraint enforces referential integrity between the values of the two tables. In this case, any employees that are added to the EMPLOYEE table must have a department number that can be found in the DEPARTMENT table.

The delete rule for the referential constraint in the employee table is NO ACTION, which means that a department cannot be deleted from the DEPARTMENT table if there are any employees in that department.

Although the previous examples use the CREATE TABLE statement to add a referential constraint, the ALTER TABLE statement can also be used. See “Altering a Table” on page 101.

Another example: The same table definitions are used as those in the previous example. Also, the DEPARTMENT table is created before the EMPLOYEE table. Each department has a manager, and that manager is listed in the EMPLOYEE table. MGRNO of the DEPARTMENT table is actually a foreign key of the EMPLOYEE table. Because of this referential cycle, this constraint poses a slight problem. You could add a foreign key later (see “Adding Primary and Foreign Keys” on page 102). You could also use the CREATE SCHEMA statement to create both the EMPLOYEE and DEPARTMENT tables at the same time (see the example in the *SQL Reference*).

FOREIGN KEY Clause: A foreign key references a primary key or a unique key in the same or another table. A foreign key assignment indicates that referential integrity is to be maintained according to the specified referential constraints. You define a foreign

key with the FOREIGN KEY clause in the CREATE TABLE or ALTER TABLE statement.

The number of columns in the foreign key must be equal to the number of columns in the corresponding primary or unique constraint (called a parent key) of the parent table. In addition, corresponding parts of the key column definitions must have the same data types and lengths. The foreign key can be assigned a *constraint name*. If you do not assign a name, one is automatically assigned. For ease of use, it is recommended that you assign a *constraint name* and do not use the system-generated name.

The value of a composite foreign key matches the value of a parent key **if** the value of each column of the foreign key is equal to the value of the corresponding column of the parent key. A foreign key containing null values cannot match the values of a parent key, since a parent key by definition can have no null values. However, a null foreign key value is always valid, regardless of the value of any of its non-null parts.

The following rules apply to foreign key definitions:

- A table can have many foreign keys
- A foreign key is nullable if any part is nullable
- A foreign key value is null if any part is null.

REFERENCES Clause: The REFERENCES clause identifies the parent table in a relationship, and defines the necessary constraints. You can include it in a column definition or as a separate clause accompanying the FOREIGN KEY clause, in either the CREATE TABLE or ALTER TABLE statements.

If you specify the REFERENCES clause as a column constraint, an implicit column list is composed of the column name or names that are listed. Remember that multiple columns can have separate REFERENCES clauses, and that a single column can have more than one.

Included in the REFERENCES clause is the delete rule. In our example, the ON DELETE NO ACTION rule is used, which states that no department can be deleted if there are employees assigned to it. Other delete rules include ON DELETE CASCADE, ON DELETE SET NULL, and ON DELETE RESTRICT. See “DELETE Rules” on page 20.

Implications for Utility Operations: The LOAD utility will turn off constraint checking for self-referencing and dependent tables, placing these tables into check pending state. After the LOAD utility has completed, you will need to turn on the constraint checking for all tables for which it was turned off. For example, if the DEPARTMENT and EMPLOYEE tables are the only tables that have been placed in check pending state, you can execute the following command:

```
SET CONSTRAINTS FOR DEPARTMENT, EMPLOYEE IMMEDIATE CHECKED
```

The IMPORT utility is affected by referential constraints in the following ways:

- The REPLACE and REPLACE CREATE functions are not allowed if the object table has any dependents other than itself.

To use these functions, first drop all foreign keys in which the table is a parent. When the import is complete, re-create the foreign keys with the ALTER TABLE statement.

- The success of importing into a table with self-referencing constraints depends on the order in which the rows are imported.

Defining a Table Check Constraint: A table check constraint specifies a search condition that is enforced for each row of the table on which the table check constraint is defined. You create a table check constraint on a table by associating a check-constraint definition with the table when the table is created or altered. This constraint is automatically activated when an INSERT or UPDATE statement modifies the data in the table. A table check constraint has no effect on a DELETE or SELECT statement.

A constraint name cannot be the same as any other constraint specified within the same CREATE TABLE statement. If you do not specify a constraint name, the system generates an 18-character unique identifier for the constraint.

A table check constraint is used to enforce data integrity rules not covered by key uniqueness or a referential integrity constraint. In some cases, a table check constraint can be used to implement domain checking. The following constraint issued on the CREATE TABLE statement ensures that the start date for every activity is not after the end date for the same activity:

```
CREATE TABLE EMP_ACT
   (EMPNO      CHAR(6)      NOT NULL,
    PROJNO     CHAR(6)      NOT NULL,
    ACTNO      SMALLINT     NOT NULL,
    EMPTIME    DECIMAL(5,2),
    EMSTDATE   DATE,
    EMENDATE   DATE,
    CONSTRAINT ACTDATES CHECK(EMSTDATE <= EMENDATE) )
IN RESOURCE
```

Although the previous example uses the CREATE TABLE statement to add a table check constraint, the ALTER TABLE statement can also be used. See “Altering a Table” on page 101.

Creating a Table in Multiple Table Spaces

Data, index, and long column data can be stored in the same table space as the table or in a different table space only for DMS. The following example shows how the EMP_PHOTO table could be created to store the different parts of the table in different table spaces:

```
CREATE TABLE EMP_PHOTO
   (EMPNO        CHAR(6)      NOT NULL,
    PHOTO_FORMAT VARCHAR(10)  NOT NULL,
    PICTURE      BLOB(100K) )
IN RESOURCE
INDEX IN RESOURCE_INDEXES
LONG  IN RESOURCE_PHOTO
```

This example will cause the EMP_PHOTO data to be stored as follows:

- Indexes created for the EMP_PHOTO table will be stored in the RESOURCES_INDEXES table space
- Data for the PICTURE column will be stored in the RESOURCE_PHOTO table space
- Data for the EMPNO and PHOTO_FORMAT columns will be stored in the RESOURCE table space.

See “Table Space Design Considerations” on page 46 for additional considerations on the use of multiple DMS table spaces for a single table.

See the *SQL Reference* for more information.

Creating a Table in a Partitioned Database

Before creating a table that will be physically divided or partitioned, you need to consider the following:

- Table spaces can span more than one database partition. The number of partitions they scan depends on the number of partitions in a nodegroup.
- Tables can be collocated by being placed in the same table space or by being placed in another table space that, together with the first table space, is associated with the same nodegroup. For more information, see “Table Collocation” on page 37.

One additional option exists when creating a table in a partitioned database environment: the *partitioning key*. A partitioning key is a key that is part of the definition of a table. It determines the partition on which each row of data is stored.

It is important to select an appropriate partitioning key because *it cannot be changed later*. Furthermore, any unique indexes (and therefore unique or primary keys) must be defined as a superset of the partitioning key. That is, if a partitioning key is defined, unique keys and primary keys must include all of the same columns as the partitioning key (they may have more columns).

If you do not specify the partitioning key explicitly, the following defaults are used. *Ensure that the default partitioning key is appropriate.*

- If a primary key is specified in the CREATE TABLE statement, the first column of the primary key is used as the partitioning key.
- If there is no primary key, the first column that is not a long field is used.
- If no columns satisfy the requirements for a default partitioning key, the table is created without one (this is allowed only in single-partition nodegroups).

Following is an example:

```
CREATE TABLE MIXREC (MIX_CNTL INTEGER NOT NULL,
                     MIX_DESC CHAR(20) NOT NULL,
                     MIX_CHR  CHAR(9) NOT NULL,
                     MIX_INT INTEGER NOT NULL,
                     MIX_INTS SMALLINT NOT NULL,
                     MIX_DEC DECIMAL NOT NULL,
                     MIX_FLT FLOAT NOT NULL,
                     MIX_DATE DATE NOT NULL,
                     MIX_TIME TIME NOT NULL,
                     MIX_TMSTMP TIMESTAMP NOT NULL)
                     IN MIXTS12
                     PARTITIONING KEY (MIX_INT) USING HASHING
```

In the preceding example, the table space is `MIXTS12` and the partitioning key is `MIX_INT`. If the partitioning key is not specified explicitly, it is `MIX_CNTL`. (If no primary key is specified and no partitioning key is defined, the partitioning key is the first non-long column in the list.)

A row of a table, and all information about that row, always resides on the same database partition.

The size limit for one partition of a table is 64 GB, or the available disk space, whichever is smaller. The size of the table can be as large as 64 GB (or the available disk space) times the number of database partitions.

Creating a Trigger

A trigger defines a set of actions that are executed in conjunction with, or triggered by, an INSERT, UPDATE, or DELETE clause on a specified base table. Some uses of triggers are to:

- Validate input data
- Generate a value for a newly-inserted row
- Read from other tables for cross-referencing purposes
- Write to other tables for audit-trail purposes

You can use triggers to support general forms of integrity or business rules. For example, a trigger can check a customer's credit limit before an order is accepted or update a summary data table.

The benefits of using a trigger are:

- Faster application development: Because a trigger is stored in the database, you do not have to code the actions it does in every application.
- Easier maintenance: Once a trigger is defined, it is automatically invoked when the table that it is created on is accessed.
- Global enforcement of business rules: If a business policy changes, you only need to change the trigger and not each application program.

The following SQL statement creates a trigger that increases the number of employees each time a new person is hired, by adding 1 to the number of employees (NBEMP)

column in the COMPANY_STATS table each time a row is added to the EMPLOYEE table.

```
CREATE TRIGGER NEW_HIRED
   AFTER INSERT ON EMPLOYEE
   FOR EACH ROW MODE DB2SQL
   UPDATE COMPANY_STATS SET NBEMP = NBEMP+1;
```

A trigger body can include one or more of the following SQL statements: INSERT, searched UPDATE, searched DELETE, full-selects, SET transition-variable, and SIGNAL SQLSTATE. See the *SQL Reference* and the *Embedded SQL Programming Guide* for information about creating and using triggers. The trigger can be activated before or after the INSERT, UPDATE, or DELETE statement to which it refers.

Trigger Dependencies

All dependencies of a trigger on some other object are recorded in the SYSCAT.TRIGDEP catalog. A trigger can depend on the following objects:

- The base table on which the trigger is defined, or an alias representing the base table
- Any tables referenced in the trigger action
- Any views referenced in the triggered action
- Any function instances used in the triggered action
- Any alias references in the triggered action
- Authorization privileges required by the trigger

If one of these objects is dropped, the trigger becomes inoperative but its definition is retained in the catalog. To revalidate this trigger, you must retrieve its definition from the catalog and submit a new CREATE TRIGGER statement.

If a trigger is dropped, its description is deleted from the SYSCAT.TRIGGERS catalog view and all of its dependencies are deleted from the SYSCAT.TRIGDEP catalog view. All packages having UPDATE, INSERT, or DELETE dependencies on the trigger are invalidated.

If the dependent object is a view and it is made inoperative, the trigger is also marked inoperative. Any packages dependent on triggers that have been marked inoperative are invalidated. (For more information, see “Statement Dependencies When Changing Objects” on page 108.)

Creating a User-Defined Function (UDF)

User-defined functions (UDFs) extend and add to the support provided by built-in functions of SQL, and can be used wherever a built-in function can be used. You can create UDFs as either:

- An external function, which is written in a programming language.
- A sourced function, whose implementation is inherited from some other existing function.

There are three types of UDFs:

Scalar Returns a single-valued answer each time it is called. For example, the built-in function SUBSTR() is a scalar function. Scalar UDFs can be either external or sourced.

Column Returns a single-valued answer from a set of like values (a column). It is also sometimes called an aggregating function in DB2. An example of a column function is the built-in function AVG(). An external column UDF cannot be defined to DB2, but a column UDF which is sourced upon one of the built-in column functions can be defined. This is useful for distinct types.

For example, if there is a distinct type SHOESIZE defined with base type INTEGER, a UDF AVG(SHOESIZE) which is sourced on the built-in function AVG(INTEGER) could be defined, and it would be a column function.

Table Returns a table to the SQL statement which references it. Table functions may only be referenced in the FROM clause of a SELECT statement. Such a function can be used to apply SQL language processing power to data which is not DB2 data, or to convert such data into a DB2 table.

For example, table functions can take a file and convert it to a table, tabularize sample data from the World Wide Web, or access a Lotus Notes database and return information such as the date, sender, and text of mail messages. This information can be joined with other tables in the database.

A table function can only be an external function. It cannot be a sourced function.

Information about existing UDFs is recorded in the SYSCAT.FUNCTIONS and SYSCAT.FUNCPARMS catalog views. The system catalog does **not** contain the executable code for the UDF. (Therefore, when creating your backup and recovery plans you should consider how you will manage your UDF executables.)

Statistics about the performance of UDFs are important when compiling SQL statements. For information about how to update UDF statistics in the system catalog, see “Updating Statistics for User-Defined Functions” on page 335.

A UDF cannot be dropped if a view, trigger, table check constraint, or another UDF is dependent on it. If a UDF is dropped, packages that are dependent on it are marked inoperative. (For more information, see “Statement Dependencies When Changing Objects” on page 108.)

For details on using the CREATE FUNCTION statement to write a UDF to suit your specific application, see the *Embedded SQL Programming Guide*. See the *SQL Reference* for details on UDF syntax.

Creating a User-Defined Type (UDT)

A user-defined type (UDT) is a distinct type derived from an existing type, such as an integer, decimal, or character type. UDTs support strong typing, which means that they share the same representations as the type they are derived from, but by definition the derived type and the source type are incompatible. They cannot be compared directly to each other.

Instances of the same distinct type can be compared to each other, if the WITH COMPARISONS clause is specified on the CREATE DISTINCT TYPE statement. Instances of UDTs cannot be used as arguments of functions or operands of operations that were defined on the source type. Similarly, the source type cannot be used in arguments or operands that were defined to use a UDT.

The SYSCAT.DATATYPES catalog view allows you to see the UDTs that have been defined for your database. This catalog view also shows you the data types defined by the database manager when the database was created. For a complete list of all data types, see the *SQL Reference*.

The following SQL statement creates the UDT `t_educ` as a smallint:

```
CREATE DISTINCT TYPE T_EDUC AS SMALLINT WITH COMPARISONS
```

Once you have created a UDT, you can use it to define columns in a CREATE TABLE statement:

```
CREATE TABLE EMPLOYEE
   (EMPNO      CHAR(6)      NOT NULL,
    FIRSTNME   VARCHAR(12)  NOT NULL,
    LASTNAME   VARCHAR(15)  NOT NULL,
    WORKDEPT   CHAR(3),
    PHONENO    CHAR(4),
    PHOTO      BLOB(10M)    NOT NULL,
    EDLEVEL    T_EDUC)
IN RESOURCE
```

A UDT cannot be used as an argument for most of the system-provided, or built-in, functions. User-defined functions must be provided to enable these and other operations. For more information about creating and using user-defined functions, see the *SQL Reference* and the *Embedded SQL Programming Guide*.

You can drop a UDT only if:

- It is **not** used in a column definition for an existing table.
- It is **not** used in a UDF function that cannot be dropped. A UDF cannot be dropped if a view, trigger, table check constraint, or another UDF is dependent on it.

When a UDT is dropped, any functions that are dependent on it are also dropped.

Creating a View

Views are derived from one or more base tables or views, and can be used interchangeably with base tables when retrieving data. When changes are made to the data shown in a view, the data is changed in the table itself.

A view can be created to limit access to sensitive data, while allowing more general access to other data. For example, the EMPLOYEE table may have salary information in it, which should not be made available to everyone. The employee's phone number, however, should be generally accessible. In this case, a view could be created from the LASTNAME and PHONENO columns only. Access to the view could be granted to PUBLIC, while access to the entire EMPLOYEE table could be restricted to those who have the authorization to see salary information. For information about *read-only* views, see the *SQL Reference* manual.

With a view, you can make a subset of table data available to an application program and validate data that is to be inserted or updated. A view can have column names that are different from the names of corresponding columns in the original tables.

The use of views provides flexibility in the way your programs and end-user queries can look at the table data.

The following SQL statement creates a view on the EMPLOYEE table that lists all employees in Department A00 with their employee and telephone numbers:

```
CREATE VIEW EMP_VIEW (DA00NAME, DA00NUM, PHONENO)
   AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

The first line of this statement names the view and defines its columns. The name EMP_VIEW must be unique within its schema in SYSCAT.TABLES. The view name appears as a table name although it contains no data. The view will have three columns called DA00NAME, DA00NUM, and PHONENO, which correspond to the columns LASTNAME, EMPNO, and PHONENO from the EMPLOYEE table. The column names listed apply one-to-one to the select list of the SELECT statement. If column names are not specified, the view uses the same names as the columns of the result table of the SELECT statement.

The second line is a SELECT statement that describes which values are to be selected from the database. It may include the clauses ALL, DISTINCT, FROM, WHERE, GROUP BY, and HAVING. The name or names of the data objects from which to select columns for the view must follow the FROM clause.

The WITH CHECK OPTION clause indicates that any updated or inserted row to the view must be checked against the view definition, and rejected if it does not conform. This enhances data integrity but requires additional processing. If this clause is omitted, inserts and updates are not checked against the view definition.

The following SQL statement creates the same view on the EMPLOYEE table using the SELECT AS clause:

```
CREATE VIEW EMP_VIEW
   SELECT LASTNAME AS DA00NAME,
          EMPNO AS DA00NUM,
          PHONENO
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

Once a view is defined, it cannot be changed. To modify a view definition, you must drop and re-create it.

You can create a view that uses a UDF in its definition. However, to update this view so that it contains the latest functions, you must drop it and then re-create it. If a view is dependent on a UDF, that function cannot be dropped.

The following SQL statement creates a view with a function in its definition:

```
CREATE VIEW EMPLOYEE_PENSION (NAME, PENSION)
  AS SELECT NAME, PENSION(HIREDATE,BIRTHDATE,SALARY,BONUS)
  FROM EMPLOYEE
```

The UDF function PENSION calculates the current pension an employee is eligible to receive, based on a formula involving their HIREDATE, BIRTHDATE, SALARY, and BONUS.

In addition to using views as described above, a view can also be used to:

- Alter a table without affecting application programs
- Sum the values in a column, select the maximum values, or average the values.

An alternative to creating a view is to use a nested or common table expression to reduce catalog lookup and improve performance. See the *SQL Reference* for more information about common table expressions.

Creating an Alias

An alias is an indirect method of referencing a table or view, so that an SQL statement can be independent of the qualified name of that table or view. Only the alias definition must be changed if the table or view name changes. An alias can be created on another alias. An alias can be used in a view or trigger definition and in any SQL statement, except for table check-constraint definitions, in which an existing table or view name can be referenced.

The alias is replaced at statement compilation time by the table or view name. If the alias or alias chain cannot be resolved to a table or view name, an error results. For example, if WORKERS is an alias for EMPLOYEE, then at compilation time:

```
SELECT * FROM WORKERS
```

becomes in effect

```
SELECT * FROM EMPLOYEE
```

An alias name can be used wherever an existing table name can be used, and can refer to another alias if no circular or repetitive references are made along the chain of aliases.

The following SQL statement creates an alias WORKERS for the EMPLOYEE table:

```
CREATE ALIAS WORKERS FOR EMPLOYEE
```

The alias name cannot be the same as an existing table, view, or alias, and can only refer to a table within the same database. The name of a table or view used in a CREATE TABLE or CREATE VIEW statement cannot be the same as an alias name in the same schema.

You do not require special authority to create an alias, unless the alias is in a schema other than the one owned by your current authorization ID, in which case DBADM authority is required.

An alias can be defined for a table, view, or alias that does not exist at the time of definition. However, it must exist when an SQL statement containing the alias is compiled.

When an alias, or the object to which an alias refers, is dropped, all packages dependent on the alias are marked invalid and all views and triggers dependent on the alias are marked inoperative.

Note: DB2 for MVS/ESA employs two distinct concepts of aliases: ALIAS and SYNONYM. These two concepts differ from DB2 Universal Database as follows:

- ALIASes in DB2 for MVS/ESA:
  - Require their creator to have special authority or privilege
  - Cannot reference other aliases.
- SYNONYMs in DB2 for MVS/ESA:
  - Can only be used by their creator
  - Are always unqualified
  - Are dropped when a referenced table is dropped
  - Do not share namespace with tables or views.

Creating an Index

An index is a list of the locations of rows, sorted by the contents of one or more specified columns. Indexes are typically used to speed up access to a table. However, they can also serve a logical data design purpose. For example, a *unique index* does not allow entry of duplicate values in the columns, thereby guaranteeing that no two rows of a table are the same. Indexes can also be created to specify ascending or descending order of the values in a column.

An index is defined by columns in the base table. It can be defined by the creator of a table, or by a user who knows that certain columns require direct access. Up to 16 columns can be specified for an index. A primary index key is automatically created on the primary key, unless a user-defined index already exists.

Any number of indexes can be defined on a particular base table and they can have a beneficial effect on the performance of queries. However, the more indexes there are, the more the database manager must modify during update, delete, and insert operations. Creating a large number of indexes for a table that receives many updates can slow down processing of requests. Therefore, use indexes only where a clear advantage for frequent access exists.

An *index key* is a column or collection of columns on which an index is defined, and determines the usefulness of an index. Although the order of the columns making up an index key does not make a difference to index key creation, it may make a difference to the optimizer when it is deciding whether or not to use an index.

If the table being indexed is empty, an index is still created, but no index entries are made until the table is loaded or rows are inserted. If the table is not empty, the database manager makes the index entries while processing the CREATE INDEX statement.

Indexes for tables in a partitioned database are built using the same CREATE INDEX statement. They are partitioned based on the partitioning key of the table. An index on a table is made of the local indexes in that table on each node in the nodegroup. Note that unique indexes defined in a multiple partition environment must be a superset of the partitioning key.

Performance Tip: Create your indexes before using the LOAD utility if you are going to carry out the following series of tasks:

- Create Table
- Load Table
- Create Index
- Perform RUNSTATS

You should consider ordering the execution of tasks in the following way:

1. Create the table
2. Create the index
3. Load the table with the `index create` and `statistics update` options requested.

For more information on LOAD performance improvements, see “System Catalog Tables” on page 27.

Indexes are maintained after they are created. Subsequently, when application programs use a key value to randomly access and process rows in a table, the index based on that key value can be used to access rows directly. This is important, because the physical storage of rows in a base table is not ordered. When a row is inserted, it is placed in the most convenient storage location that can accommodate it. When searching for rows of a table that meet a particular selection condition and the table has no indexes, the entire table is scanned. An index optimizes data retrieval without performing a lengthy sequential search.

The data for your indexes can be stored in the same table space as your table data, or in a separate table space containing index data. The table space used to store the

index data is determined when the table is created (see “Creating a Table in Multiple Table Spaces” on page 87).

The following two sections “Using an Index” and “Using the CREATE INDEX Statement” provide more information on creating an index.

Using an Index

An index is never directly used by an application program. The decision on whether to use an index and which of the potentially available indexes to use is the responsibility of the optimizer.

The best index on a table is one that:

- Uses high-speed disks
- Is highly-clustered
- Is made up of only a few narrow columns

For a detailed discussion of how an index can be beneficial, see “Index Scan Concepts” on page 349.

Using the CREATE INDEX Statement

You can create an index that will allow duplicates (a non-unique index) to enable efficient retrieval by columns other than the primary key, and allow duplicate values to exist in the indexed column or columns.

The following SQL statement creates a non-unique index called LNAME from the LASTNAME column on the EMPLOYEE table, sorted in ascending order:

```
CREATE INDEX LNAME ON EMPLOYEE (LASTNAME ASC)
```

The following SQL statement creates a unique index on the phone number column:

```
CREATE UNIQUE INDEX PH ON EMPLOYEE (PHONENO DESC)
```

A unique index ensures that no duplicate values exist in the indexed column or columns. The constraint is enforced at the end of the SQL statement that updates rows or inserts new rows. This type of index cannot be created if the set of one or more columns already has duplicate values.

The keyword ASC puts the index entries in ascending order by column, while DESC puts them in descending order by column. The default is ascending order.

In multiple partition databases, unique indexes must be defined as supersets of the partitioning key.

For intra-partition parallelism, `index create` performance is improved by using multiple processors for the scanning and sorting of data that is performed during index creation. The use of multiple processors is enabled by setting `intra_parallel` to YES(1) or ANY(-1). The number of processors used during `index create` is determined by the system and is not affected by the configuration parameters `dft_degree` or `max_querydegree`, by the application runtime degree, or by the SQL statement

compilation degree. If the database configuration parameter `index sort` is NO, then `index create` will not use multiple processors.

Before Altering a Database

After a database design has been implemented, but before changing the database design, you should consider the following:

- Changing Logical and Physical Design Characteristics
- Changing Environment Variables and the Profile Registry Variables
- Changing the Node Configuration File
- Changing the Database Configuration File

Changing Logical and Physical Design Characteristics

Before you make changes affecting the entire database, you should review all the logical and physical design decisions. For example, when altering a table space, you should review your design decision regarding the use of SMS or DMS storage types. (See "Designing and Choosing Table Spaces" on page 38.)

Changing Environment Variables and the Profile Registry Variables

You must consider which environment variables (if any) need to be changed on your particular operating system. If any environment variables are changed, you need to restart the system for the new environment variables to take effect. Review whether you should reset the profile registry variables in the Global Profile registry before altering your database. You can then reset the profile registry values to those that are best suited to the new database environment. If only profile registry values have been changed, the system does not need to be restarted.

Changing the Node Configuration File

If you are planning changes to any nodegroups, you should review the contents of the node configuration file, `db2nodes.cfg`. When adding a new database partition to a nodegroup, you should first ensure that the database partition is one of those listed in the node configuration file. The database partition to be added to a nodegroup may already be part of another nodegroup and therefore already be in the node configuration file. If it is not, you must stop DB2, change the file by adding the new database partition information and then restart DB2. The new entry in the node configuration file takes effect the next time that DB2 is started. You should not remove the database partition entry in the node configuration file when removing a node from a nodegroup.

Changing the Database Configuration File

If you are planning changes to the database, you should review the values for the configuration parameters. Some of the values can be adjusted as part of the changes made to the database. However, just as with the initial setting of the configuration parameters, you may want to consider doing some benchmark testing to achieve optimal performance for your database. For more information on benchmark testing, see Chapter 18, "Benchmark Testing" on page 447.

Altering a Database

You need to do some tasks before making a change to your database. These tasks include:

- Altering a Nodegroup
- Dropping a Database

Altering a Nodegroup

You can add nodes to a nodegroup, or you can reorganize existing nodes to create a new nodegroup.

When you add a node to a nodegroup, you do *not* have to drop and re-create the tables and table spaces in the new nodegroup. To add a node to a nodegroup:

1. Add the nodename to the configuration file `db2nodes.cfg`.
2. Run the ADD NODE command to create database partitions on the new node for all databases currently defined. (See the *Command Reference* for information.)

(To drop nodes, run the DROP NODE command and then remove the nodename entries from the `db2nodes.cfg` file.)

Once you add or drop nodes, you must redistribute the current data across the new set of nodes in the nodegroup. To do this, use the REDISTRIBUTE NODEGROUP command. For information, see Chapter 16, “Redistributing Data Across Database Partitions” on page 435 and the *Command Reference*.

Dropping a Database

Although some of the objects in a database can be altered, the database itself cannot be altered: it must be dropped and re-created. Dropping a database can have far-reaching effects, because this action deletes all its objects, containers, and associated files. The dropped database is uncataloged in the database directories.

The following command deletes the database SAMPLE:

```
DROP DATABASE SAMPLE
```

Altering a Table Space

When you create a database, you create at least three table spaces: one catalog table space (SYSCATSPACE); one user table space (default name is USERSPACE1); and one temporary table space (whose default name is TEMPSPACE1). You must keep at least one of each of these table spaces, And can add additional user and temporary table spaces if you wish. Note that you cannot drop the catalog table space, SYSCATSPACE; and there must always be at least one temporary table space.

This section discusses how to change table spaces as follows:

- “Adding a Container to a DMS Table Space” on page 100
- “Dropping a User Table Space” on page 100
- “Altering the Temporary Table Space” on page 100

Adding a Container to a DMS Table Space

You can increase the size of a DMS table space (that is, one created with the MANAGED BY DATABASE clause) by adding one or more containers to the table space.

The following example illustrates how to add two new device containers (each with 40 000 pages) to a table space on a UNIX-based system:

```
ALTER TABLESPACE RESOURCE
  ADD (DEVICE '/dev/rhd9'  10000,
       DEVICE  '/dev/rhd10' 10000)
```

The contents of the table space are re-balanced across all containers. Access to the table space is not restricted during the re-balancing. If you need to add more than one container, you should add them at the same time.

Note that the ALTER TABLESPACE statement allows you to change other properties of the table space that can affect performance. For more information, see “Table Space Impact on Query Optimization” on page 304.

Dropping a User Table Space

When you drop a user table space, you delete all the data in that table space, free the containers, remove the catalog entries, and all objects defined in the table space are either dropped or marked as invalid.

You cannot drop a table space if a table stores at least one of its parts in it and one or more of its parts in another table space. The table must be dropped first.

The following SQL statement drops the table space ACCOUNTING:

```
DROP TABLESPACE ACCOUNTING
```

For information on SQL statements, see the *SQL Reference*.

You can reuse the containers in an empty table space by dropping the table space, but you must COMMIT the DROP TABLESPACE command, or have had AUTOCOMMIT on, before attempting to reuse the containers.

Altering the Temporary Table Space

You cannot drop the temporary table space, because the database must always have at least one temporary table space. If you wish to change the specifications of this table space, you must add a new temporary table space first and then drop the old temporary table space.

The following SQL statement creates a new temporary table space called TEMPSPACE2:

```
CREATE TEMPORARY TABLESPACE TEMPSPACE2
   MANAGED BY SYSTEM USING ('d')
```

Once TEMPSPACE2 is created, you can then drop the original temporary table space TEMPSPACE1 with the command:

```
DROP TABLESPACE TEMPSPACE1
```

You can reuse the containers in an empty table space by dropping the table space, but you must COMMIT the DROP TABLESPACE command, or have had AUTOCOMMIT on, before attempting to reuse the containers.

Dropping a Schema

Before dropping a schema, all objects that were in that schema must be dropped themselves or moved to another schema. To ensure that there are no objects in the schema when executing the DROP statement, use the RESTRICT attribute. The schema name must be in the catalog when attempting the DROP statement; otherwise an error is returned. In the following example, the schema "joeschma" is dropped:

```
DROP SCHEMA joeschma RESTRICT
```

Altering a Table

You should perform one or more of the following tasks when you modify a table as a result of a table design. These tasks include:

- Adding Columns to an Existing Table
- Altering a Constraint
- Adding a Constraint
- Dropping a Constraint
- Renaming an Existing Table
- Dropping a Table
- Changing Partitioning Keys

Note that you cannot alter triggers for tables; you must drop any trigger that is no longer appropriate (see “Dropping a Trigger” on page 106), and add its replacement (see “Creating a Trigger” on page 89).

Adding Columns to an Existing Table

When a new column is added to an existing table, only the table description in the system catalog is modified, so access time to the table is not affected immediately. Existing records are not physically altered until they are modified using an UPDATE statement. When retrieving an existing row from the table, a null or default value is provided for the new column, depending on how the new column was defined. Columns that are added after a table is created cannot be defined as NOT NULL: they must be defined as either NOT NULL WITH DEFAULT or as nullable.

Columns can be added with an SQL statement. The following statement uses the ALTER TABLE statement to add three columns to the EMPLOYEE table:

```
ALTER TABLE EMPLOYEE
   ADD MIDINIT CHAR(1)   NOT NULL WITH DEFAULT
   ADD HIREDATE DATE
   ADD WORKDEPT CHAR(3)
```

A column definition includes a column name, data type, and any necessary constraints. In addition to adding columns to a table, the ALTER TABLE statement can be used to add or drop a primary or foreign key and to add or drop a table check constraint definition. For more information about the ALTER TABLE statement, see the *SQL Reference* manual.

Altering a Constraint

You can only alter constraints by dropping them and then adding new ones to take their place. For more information, see:

- "Adding a Constraint"
- "Dropping a Constraint" on page 103

For more information on constraints, see "Defining Constraints" on page 83.

Adding a Constraint

You add constraints with the ALTER TABLE statement. For more information on this statement, including its syntax, see the *SQL Reference* manual.

For more information on constraints, see "Defining Constraints" on page 83.

Adding a Unique Constraint: Unique constraints can be added to an existing table. The constraint name cannot be the same as any other constraint specified within the ALTER TABLE statement, and must be unique within the table (this includes the names of any referential integrity constraints that are defined). Existing data is checked against the new condition before the statement succeeds.

The following SQL statement adds a unique constraint to the EMPLOYEE table that represents a new way to uniquely identify employees in the table:

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT NEWID UNIQUE(EMPNO,HIREDATE)
```

Adding Primary and Foreign Keys: The following examples show the ALTER TABLE statement to add primary keys and foreign keys to a table:

```
ALTER TABLE PROJECT
  ADD CONSTRAINT PROJECT_KEY
      PRIMARY KEY (PROJNO)
ALTER TABLE EMP_ACT
  ADD CONSTRAINT ACTIVITY_KEY
      PRIMARY KEY (EMPNO, PROJNO, ACTNO)
  ADD CONSTRAINT ACT_EMP_REF
      FOREIGN KEY (EMPNO)
      REFERENCES EMPLOYEE
      ON DELETE  RESTRICT
  ADD CONSTRAINT ACT_PROJ_REF
      FOREIGN KEY (PROJNO)
      REFERENCES PROJECT
      ON DELETE CASCADE
```

To add constraints to a large table, it is more efficient to put the table into the check pending state, add the constraints, and then check the table for a consolidated list of violating rows. Use the SET CONSTRAINTS statement to explicitly set the check pending state: if the table is a parent table, check pending is implicitly set for all dependent and descendent tables.

When a foreign key is added to a table, packages and cached dynamic SQL containing the following statements may be marked as invalid:

- Statements that insert or update the table containing the foreign key
- Statements that update or delete the parent table.

See "Statement Dependencies When Changing Objects" on page 108 for information.

Adding a Table Check Constraint: Check constraints can be added to an existing table with the ALTER TABLE statement. The constraint name cannot be the same as any other constraint specified within an ALTER TABLE statement, and must be unique within the table (this includes the names of any referential integrity constraints that are defined). Existing data is checked against the new condition before the statement succeeds.

The following SQL statement adds a constraint to the EMPLOYEE table that the salary plus commission of each employee must be more than $25,000:

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT REVENUE CHECK (SALARY + COMM > 25000)
```

To add constraints to a large table, it is more efficient to put the table into the check-pending state, add the constraints, and then check the table for a consolidated list of violating rows. Use the SET CONSTRAINTS statement to explicitly set the check-pending state: if the table is a parent table, check pending is implicitly set for all dependent and descendent tables.

When a table check constraint is added, packages and cached dynamic SQL that insert or update the table may be marked as invalid. See "Statement Dependencies When Changing Objects" on page 108 for more information.

Dropping a Constraint

You drop constraints with the ALTER TABLE statement. For more information on this statement, including its syntax, see the *SQL Reference* manual.

For more information on constraints, see "Defining Constraints" on page 83.

Dropping a Unique Constraint: You can explicitly drop a unique constraint using the ALTER TABLE statement. The name of all unique constraints on a table can be found in the SYSCAT.INDEXES system catalog view.

The following SQL statement drops the unique constraint NEWID from the EMPLOYEE table:

```
ALTER TABLE EMPLOYEE
   DROP UNIQUE NEWID
```

Dropping this unique constraint invalidates any packages or cached dynamic SQL that used the constraint.

Dropping Primary and Foreign Keys: The following examples use the DROP PRIMARY KEY and DROP FOREIGN KEY clauses in the ALTER TABLE statement to drop primary keys and foreign keys on a table:

```
ALTER TABLE EMP_ACT
  DROP PRIMARY KEY
  DROP FOREIGN KEY ACT_EMP_REF
  DROP FOREIGN KEY ACT_PROJ_REF
ALTER TABLE PROJECT
  DROP PRIMARY KEY
```

For information about the ALTER TABLE statement, see the *SQL Reference* manual.

When a foreign key constraint is dropped, packages or cached dynamic SQL statements containing the following may be marked as invalid:

- Statements that insert or update the table containing the foreign key
- Statements that update or delete the parent table.

See “Statement Dependencies When Changing Objects” on page 108 for more information.

Dropping a Table Check Constraint: You can explicitly drop or change a table check constraint using the ALTER TABLE statement, or implicitly drop it as the result of a DROP TABLE statement. The name of all check constraints on a table can be found in the SYSCAT.CHECKS catalog view.

The following SQL statement drops the table check constraint REVENUE from the EMPLOYEE table:

```
ALTER TABLE EMPLOYEE
   DROP CHECK REVENUE
```

When you drop a table check constraint, all packages and cached dynamic SQL statements with INSERT or UPDATE dependencies on the table are invalidated. (See “Statement Dependencies When Changing Objects” on page 108 for more information.) To drop a table check constraint with a system-generated name, look for the name in the SYSCAT.CHECKS catalog view.

Renaming an Existing Table

You can give an existing table a new name within a schema and maintain the authorizations and indexes that were created on the original table.

The existing table cannot be referenced in any of the following:

- Views
- Triggers
- Referential constraints.

Also, there must be no check constraints within the table. Any packages or cached dynamic SQL statements dependent on the original table are invalidated. Finally, any aliases referring to the original table are not modified.

You should consider checking the appropriate system catalog tables to ensure that the table being renamed is not affected by any of these restrictions.

The SQL statement below renames the EMPLOYEE table within the COMPANY schema to EMPL:

```
RENAME TABLE COMPANY.EMPLOYEE TO EMPL
```

Packages must be re-bound if they refer to a table that has just been renamed. The packages can be implicitly re-bound if:

- Another table is renamed using the original name of the table, or
- An alias or view is created using the original name of the table.

One of these two choices must be completed before any implicit or explicit re-binding is attempted. If neither choice is made, any re-bind will fail.

For more information about the RENAME TABLE statement, see the *SQL Reference* manual.

Dropping a Table

A table can be dropped with a DROP TABLE SQL statement. The following statement drops the table called EMPLOYEE:

```
DROP TABLE EMPLOYEE
```

When a table is dropped, the row in the SYSCAT.TABLES catalog that contains information about that table is dropped, and any other objects that depend on the table are affected. For example:

- All column names are dropped.
- Indexes created on any columns of the table are dropped.
- All views based on the table are marked inoperative. (See “Recovering Inoperative Views” on page 107 for more information.)
- All privileges on the dropped table and dependent views are implicitly revoked.
- All referential constraints in which the table is a parent or dependent are dropped.
- All packages and cached dynamic SQL statements dependent on the dropped table are marked invalid, and remain so until the dependent objects are re-created. (See “Statement Dependencies When Changing Objects” on page 108 for more information.)
- An alias definition on the table is not affected, because an alias can be undefined
- All triggers dependent on the dropped table are marked inoperative.

Changing Partitioning Keys

You can only change a partitioning key on tables in single-partition nodegroups. This is done by first dropping the existing partitioning key and then creating another.

The following SQL statement drops the partitioning key MIX_INT from the MIXREC table:

```
ALTER TABLE MIXREC
   DROP PARTITIONING KEY MIX_INT
```

For more information, see the ALTER TABLE statement in the *SQL Reference* manual.

You cannot change the partitioning key of a table in a multiple database partition nodegroup. If you try to drop it, an error is returned.

The only methods to change the partitioning key of multiple database partition nodegroups are either:

- Export all of the data to a single-partition nodegroup and then follow the above instructions.
- Export all of the data, drop the table, redefine the partitioning key, and then import all of the data.

Neither of these methods are practical for large databases; it is therefore essential that you define the appropriate partitioning key before implementing the design of large databases.

Dropping a Trigger

A trigger object can be dropped using the DROP statement, but this procedure will cause dependent packages to be marked invalid, as follows:

- If an update trigger without an explicit column list is dropped, then packages with an update usage on the target table are invalidated.
- If an update trigger with a column list is dropped, then packages with update usage on the target table are only invalidated if the package also had an update usage on at least one column in the `column-name` list of the CREATE TRIGGER statement.
- If an insert trigger is dropped, packages that have an insert usage on the target table are invalidated.
- If a delete trigger is dropped, packages that have a delete usage on the target table are invalidated.

A package remains invalid until the application program is explicitly bound or rebound, or it is run and the database manager automatically rebinds it.

Dropping a User-Defined Function (UDF)

A user-defined function (UDF) can be dropped using the DROP statement. Functions implicitly generated by the CREATE DISTINCT TYPE statement cannot be dropped. It is not possible to drop a function that is in either the SYSIBM schema or the SYSFUN schema.

Other objects can be dependent on a function. All such dependencies must be removed before the function can be dropped, with the exception of packages which are marked inoperative. Such a package is not implicitly rebound. It must either be rebound using the BIND or REBIND commands or it must be prepared by use of the PREP command. See the *Command Reference* manual for more information on these commands.

Dropping a UDF invalidates any packages or cached dynamic SQL statements that used it.

Dropping a User-Defined Type

You can drop a user-defined type (UDT) using the DROP DISTINCT TYPE statement. You cannot drop a UDT if it is used in a column definition for an existing table. The database manager will attempt to drop all functions that are dependent on this distinct type. If the UDF cannot be dropped, the UDT cannot be dropped. A UDF cannot be dropped if a view, trigger, table check constraint, or another UDF is dependent on it. Dropping a UDT invalidates any packages or cached dynamic SQL statements that used it.

For more information about the user-defined types, see the *SQL Reference* and *Embedded SQL Programming Guide* manuals.

Dropping a View

You may not change a view definition; the view must be dropped and re-created.

The following example shows how to drop the EMP_VIEW:

```
DROP VIEW EMP_VIEW
```

Any views that are dependent on the view being dropped will be made inoperative. (See “Recovering Inoperative Views” for more information.)

Other database objects such as tables and indexes will not be affected although packages and cached dynamic statements are marked invalid. See “Statement Dependencies When Changing Objects” on page 108 for more information.

For more information on dropping and creating views, see the *SQL Reference* manual.

Recovering Inoperative Views

When an object is dropped, views can become *inoperative* if they are dependent on that object.

The following steps can help you recover an inoperative view:

- Determine the SQL statement that was initially used to create the view. You can obtain this information from the TEXT column of the SYSCAT.VIEW catalog view.
- Re-create the view by using the CREATE VIEW statement with the same view name and same definition.
- Use the GRANT statement to re-grant all privileges that were previously granted on the view. (Note that all privileges granted on the inoperative view are revoked.)

If you do not want to recover an inoperative view, you can explicitly drop it with the DROP VIEW statement, or you can create a new view with the same name but a different definition.

An inoperative view only has entries in the SYSCAT.TABLES and SYSCAT.VIEWS catalog views; all entries in the SYSCAT.VIEWDEP, SYSCAT.TABAUTH, SYSCAT.COLUMNS and SYSCAT.COLAUTH catalog views are removed.

Dropping an Index

You cannot change any clause of an index definition; you must drop the index and create it again. (Dropping an index does not cause any other objects to be dropped but may cause some packages to be invalidated.)

The following SQL statement drops the index called PH:

```
DROP INDEX PH
```

A primary key or unique key index cannot be explicitly dropped. You must use one of the following methods to drop it:

- If the primary index or unique constraint was created automatically for the primary key or unique key, dropping the primary key or unique key will cause the index to be dropped. Dropping is done through the ALTER TABLE statement.
- If the primary index or the unique constraint was user-defined, the primary key or unique key must be dropped first, through the ALTER TABLE statement. After the primary key or unique key is dropped, the index is no longer considered the primary index or unique index, and it can be explicitly dropped.

If you drop a non-primary or non-unique index, any packages and cached dynamic SQL statements that depend on the indexes are marked invalid. See “Statement Dependencies When Changing Objects” for more information. The application program is not affected by changes resulting from adding or dropping indexes.

Statement Dependencies When Changing Objects

Statement dependencies include package and cached dynamic SQL statements. A *package* is a database object that contains the information needed by the database manager to access data in the most efficient way for a particular application program. *Binding* is the process that creates the package the database manager needs in order to access the database when the application is executed. The *Embedded SQL Programming Guide* discusses how to create packages in detail.

Packages and cached dynamic SQL statements can be dependent on the following types of objects:

- Schemas
- Tables
- Views
- Aliases
- Indexes
- User-defined functions
- Triggers
- Referential constraints
- Table check constraints.

These objects could be explicitly referenced, for example, a table or user-defined function that is involved in an SQL SELECT statement. The objects could also be implicitly referenced, for example, a dependent table that needs to be checked to ensure that referential constraints are not violated when a row in a parent table is deleted. Packages are also dependent on the privileges which have been granted to the package creator.

If a package or cached dynamic SQL statement depends on an object and that object is dropped, the package or cached dynamic SQL statement will be placed in an "invalid" state. If the object that is dropped is a user-defined function, the package is placed in an "inoperative" state.

Packages or cached dynamic SQL statements in an "invalid" state are implicitly rebound the next time they are accessed. They can also be explicitly rebound. If a package or cached dynamic SQL statement was marked invalid because a trigger was dropped, it will be rebound **without** the trigger.

Packages or cached dynamic SQL statements in an "inoperative" state must be explicitly rebound before they can be used again. See the *Embedded SQL Programming Guide* for more information about binding and rebinding packages.

In some cases, it will not be possible to rebind the package. For example, if a table has been dropped and not re-created, the package cannot be rebound. In this case, you will need to either re-create the object or change the application so it does not use the dropped object.

In many other cases, for example if one of the constraints was dropped, it will be possible to rebind the package.

The following system catalog views help you to determine the state of a package and the package's dependencies:

- SYSCAT.PACKAGEAUTH
- SYSCAT.PACKAGEDEP
- SYSCAT.PACKAGES

For more information about object dependencies, see the DROP statement in the *SQL Reference* manual.

Chapter 4. Controlling Database Access

One of the most important responsibilities of the database administrator and the system administrator is database security. Securing your database involves several activities:

- Preventing accidental loss of data or data integrity through equipment or system malfunction. (This subject is covered in “Backing Up a Database” on page 201.)
- Preventing unauthorized access to valuable data. You must ensure that sensitive information is not accessed by those without a “need to know.”
- Preventing unauthorized persons from committing mischief through malicious deletion or tampering with data.

The following topics are discussed:

Planning for Security: Start by defining your objectives for a database access control plan, and specifying who shall have access to what and under what circumstances. Your plan should also describe how to meet these objectives by using database functions, functions of other programs, and administrative procedures.

An Overview of DB2 Security

To protect data and resources associated with a database server, DB2 uses a combination of external security services and internal access control information. To access a database server you must pass some security checks before you are given access to database data or resources. The first step in database security is called *authentication*, where the user must prove he is who he says he is. The second step is called *authorization*, where the database manager decides if the validated user is allowed to perform the requested action or access the requested data.

Authentication

Authentication of a user is completed using a security facility outside of DB2. The security facility can be part of the operating system, a separate product, or, in certain cases, not exist at all. On UNIX platforms, the security facility is in the operating system itself. DCE Security Services is a separate product that provides the security facility for a distributed environment. There are no security facilities on the Windows 95 or Windows 3.1 operating systems.

The security facility requires two items to authenticate a user: first, the user is identified to the security facility by a user ID; second, the user proves he is this identity by providing a piece of information known only to the user and the security facility; for example, a password.

Once authenticated,

- The user must then be identified to DB2 using an SQL authorization name *authid*. This name can be the same as the user ID, or a mapped value. For example, on a UNIX platform, a DB2 authid is derived by transforming to upper case letters a UNIX user ID that follows DB2 naming conventions. In another example, within the DCE Security Services product, the DB2 authid is contained in the DCE registry and is extracted from there once authentication has successfully completed.
- A list of groups in which the user is a member is obtained. Group membership may be used when authorizing the user. Groups are security facility entities that must also map to DB2 authorization names. This mapping is done in a method similar to that used for user IDs.

  DB2 will obtain a list of groups up to a maximum of 64 groups. If a user is a member of more than 64 groups, only the first 64 that map to valid DB2 authorization names will be added to the DB2 group list. No error is created when this happens, and any groups after the first 64 are ignored by DB2.

DB2 uses the security facility to authenticate users in one of two ways:

- DB2 uses your successful security system login as evidence of your identity and allows the following using that identity:
  - Use of local commands to access local data
  - Use of remote connections where the server trusts the client authentication.
- DB2 accepts a user ID and password combination and uses successful validation of this pair by the security facility as evidence of your identity and allows:
  - Use of remote connections where the server requires proof of authentication
  - Use of operations where the user wants to execute a command under an identity other than the identity used for login

Authorization

Authorization is the process whereby DB2 obtains information about an authenticated DB2 user that indicates the database operations a user may perform and what data objects may be accessed. With each user request there may be more than one authorization check depending on the objects and operations involved.

Authorization is performed using DB2 facilities. DB2 tables and configuration files are used to record the permissions associated with each authorization name. The authorization name of an authenticated user, and those of groups in which the user is a member, are compared against the recorded permissions. Based on the comparison, DB2 decides whether to allow the user the requested access.

There are two types of permissions recorded by DB2: privileges and authority levels. A *privilege* defines a single permission for an authorization name, enabling a user to create or access database resources. Privileges are stored in the database catalogs for a given database. *Authority levels* provide a method of grouping privileges and control over higher level database manager maintenance and utility operations. Database-specific authorities are stored in the database catalogs for each database; system authorities are recorded by group membership and are stored in the database manager configuration file for a given instance.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. Unless otherwise specified, group authorization names can be used anywhere authorization names are used for authorization purposes. In general, group membership is considered for dynamic SQL and non-database object authorizations (such as instance level commands and utilities) and is not considered for static SQL. Specific cases where group membership does not apply are noted throughout DB2 documentation, where applicable.

"Privileges, Authorities, and Authorization" on page 121 presents further details on these topics.

The following section ("Selecting an Authentication Method for Your Server") provides additional information about the system entry validation checking that is particularly relevant if you have remote clients accessing the database.

Selecting an Authentication Method for Your Server

Access to an instance or a database first requires that the user be *authenticated*. The *authentication type* for each instance determines how and where a user will be verified. The authentication type is stored in the database manager configuration file at the server. It is initially set when the instance is created. See "Authentication Type (authentication)" on page 572 for more information on this database manager configuration parameter. There is one authentication type per instance, which covers access to that database server and all the databases under its control.

The following authentication types are provided:

SERVER Specifies that authentication occurs on the server using local operating system security. If a user ID and password are specified during the connection or attachment attempt, they are compared to the valid user ID and password combinations at the server to determine if the user is permitted to access the instance. This is the default security mechanism.

Note: The server code detects whether a connection is local or remote. For local connections, when authentication is SERVER, a user ID and password are not required for authentication to be successful.

If the remote instance has SERVER authentication, the user ID and password must be provided by the user or retrieved by DB2 and provided

to the server for validation even though the user has already logged on to the local machine or to the domain.

CLIENT Specifies that authentication occurs on the database partition where the application is invoked using operating system security. The user ID and password specified during a connection or attachment attempt are compared with the valid user ID and password combinations on the client node to determine if the user ID is permitted access to the instance. No further authentication will take place on the database server.

If the user performs a local or client login, the user is known only to that local client workstation.

If the remote instance has CLIENT authentication, two other parameters determine the final authentication type: *trust_allclnts* and *trust_clntauth*.

CLIENT level security for TRUSTED clients only:

Trusted clients are clients that have a reliable, local security system. Specifically, all clients are trusted clients except for Macintosh, Windows 3.1, and Windows 95 operating systems.

When the authentication type of CLIENT has been selected, an additional option may be selected to protect against clients whose operating environment has no inherent security.

To protect against unsecured clients, the administrator can select Trusted Client Authentication by setting the *trust_allclnts* parameter to NO. This implies that all trusted platforms can authenticate the user on behalf of the server. Untrusted clients are authenticated on the Server and must provide a user ID and password. You use the *trust_allclnts* configuration parameter to indicate whether you are trusting clients. The default for this parameter is YES. For more information on this parameter, see “Trust All Clients (trust_allclnts)” on page 574.

Note: It is possible to trust all clients (*trust_allclnts* is YES) yet have some of those clients as those who do not have a native safe security system for authentication.

You may also want to complete authentication at the server even for trusted clients. To indicate where to validate trusted clients, you use the *trust_clntauth* configuration parameter. The default for this parameter is CLIENT. See “Trusted Clients Authentication (trust_clntauth)” on page 575 for more information on this parameter.

Note: For trusted clients only, if no user ID or password is explicitly provided when attempting to CONNECT or ATTACH, then validation of the user takes place at the client. The *trust_clntauth* parameter is only used to determine where to validate the information provided on the USER/USING clauses.

Table 20. Trusted Client Options

TRUST_ALLCLNTS	TRUST_CLNTAUTH	Trusted Client Authentication no password	Trusted Client Authentication with password	Untrusted Client Authentication password required
YES (default)	CLIENT (default)	CLIENT	CLIENT	N/A
YES (default)	SERVER	CLIENT	SERVER	N/A
NO	CLIENT (default)	CLIENT	CLIENT	SERVER
NO	SERVER	CLIENT	SERVER	SERVER

DCS Primarily used to catalog a database accessed using DB2 Connect. (Refer to the *DB2 Connect User's Guide* section on Security for more details on this topic.) When it is used to specify the authentication type for an instance in the database manager configuration file, it means the same as for authentication **SERVER**, unless the server is being accessed via the Distributed Relational Database Architecture (DRDA) Application Server (AS) architecture using the Advanced Program-To-Program Communications (APPC) protocol. In this case, using **DCS** indicates that authentication will occur at the server, but only in the APPC layer. Further authentication will not occur in the DB2 code. This value is only supported when the APPC `SECURITY` parameter for the connection is specified as SAME or PROGRAM.

DCE Specifies that the user is authenticated using DCE Security Services. For more information on DCE Security, see "Using DCE Security Services to Authenticate Users" on page 117.

Note: When DB2 Connect is part of the system environment, the authentication types have slightly different meanings. Also, here we are presenting the authentication type that is stored in the database manager configuration file for the DB2 Universal Database. In DB2 Connect, the authentication types used are those stored in the database directory. Refer to the section on Security in the *DB2 Connect User's Guide* for more details on this topic.

Note: The type of authentication you choose is only important if you have remote database clients accessing the database. Most users accessing the database through local clients are always authenticated on the same machine as the database. An exception may exist when DCE Security Services are used. For information about supporting and using remote clients, see your *Quick Beginnings* manual.

Note: Do not inadvertently lock yourself out of your instance when you are changing the authentication information, since access to the configuration file itself is protected by information in the configuration file. The following database manager configuration file parameters control access to the instance:

- AUTHENTICATION *
- SYSADM_GROUP *

- TRUST_ALLCLNTS
- TRUST_CLNTAUTH
- SYSCTRL_GROUP
- SYSMAINT_GROUP

* Indicates the two most important parameters, and those most likely to cause a problem.

There are some things that can be done to ensure this does not happen: If you do accidentally lock yourself out of the DB2 system, you have a failsafe option available on all platforms that will allow you to override the usual DB2 security checks to update the database manager configuration file using a highly privileged local operating system security user. This user **always** has the privilege to update the database manager configuration file and thereby correct the problem. However, this security bypass is restricted to a local update of the database manager configuration file. You cannot use a failsafe user remotely or for any other DB2 command. This special user is identified as follows:

- UNIX platforms: the instance owner
- NT platform: someone belonging to the local “administrators” group
- OS/2 platform: a UPM administrator
- other platforms: there is no local security on the other platforms, so all users pass local security checks anyway

Authentication Considerations for Remote Clients

When cataloging a database for remote access, the authentication type may be specified in the database directory entry.

For databases accessed using DB2 Connect: If a value is not specified, SERVER authentication is assumed.

For databases accessed remotely but not using DB2 Connect: The authentication type is not required. However, if it is not specified the client must first contact the server to obtain the value before beginning the authentication flow. If specified, authentication can begin immediately provided the value specified matches that at the server. If a mismatch is detected: DB2 attempts to recover, which may result in more flows to reconcile the difference, or in an error if DB2 cannot recover. In the case of a mismatch, the value at the server is assumed to be correct.

Partitioned Database Considerations

In a partitioned database, each partition of the database must have the same set of users and groups defined. If the definitions are not the same, the user may be authorized to do different things on different partitions. Consistency across all partitions is recommended.

Using DCE Security Services to Authenticate Users

When considering security for your distributed database environment, Distributed Computing Environment (DCE) Security Services are a good option because DCE provides:

- Centralized administration of users and passwords.
- No transmission of clear text passwords and user IDs.
- A single sign-on for users.

DB2 supports DCE default login contexts, connection login contexts, and delegated contexts. A *default login context* is established when a user does a `dce_login` on a client. Subsequent DB2 commands have access to this context and may perform user authentication without further user intervention (that is, no requirement for a user ID or password). A *connection login context* is established for a DB2 session using the user ID and password provided on CONNECT or ATTACH using the USER/USING clause. Finally, a *delegated login context* occurs when a DB2 client is used as part of a DCE server application. The DCE server application (that is also a DB2 client), receives requests from a DCE client application, from which point the original identity of the user originates. Provided the DCE client and DCE server are correctly configured to allow the DCE server to be a delegate for the DCE client, DB2 will obtain the delegated token and forward this to the DB2 server. This allows the DB2 server to use the original identity of the DCE client, rather than using the identity of the DCE server, to process requests. Information on how to establish a delegated login context can be obtained from the DCE documentation for your platform.

How to Setup a DB2 User for DCE

Users must be registered in the Distributed Computing Environment (DCE) Registry and have correct attributes before being used with DB2. See the appropriate platform-specific DCE documentation for information on how to create a DCE principal.

Each DB2 user wishing to use a DCE-authenticated server must have a DCE principal and account defined in the DCE Registry with the client flag enabled. This principal must also have an entry in its Extended Registry Attributes (ERA) section showing what authorization name will be used for this principal when it connects to a particular DCE authenticated server.

You may also wish to have user principals be members of groups in order to use group privileges in the database. Similar information in the group ERA maps the group name to a DB2 authorization name. The authorization name is a secondary authorization name but the same restrictions apply. Please refer to your DCE documentation for additional information on how to create groups and add members.

The information in the ERA maps a user's DCE principal or group name to a DB2 authorization name for a particular server DCE principal name. To use an ERA, an ERA schema indicating the format of this attribute must be defined. This needs to be done once per DCE cell and is accomplished by completing the following steps:

1. Login to DCE as a valid DCE administrator

2. Invoke `dcecp` and enter the following at the prompt:

```
> xattrschema create /.:/sec/xattrschema/db2map \
> -aclmgr {{principal r m r m } {group r m r m }} \
> -annotation {Schema entry for DB2 database access} \
> -encoding stringarray \
> -multivalued no \
> -uuid 1cbe84ca-9df3-11cf-84cd-02608c2cd17b
```

This creates the Extended Registry Attribute `db2map`.

To view this mapping, issue the following command at the dcecp prompt:

```
> xattrschema show /.:/sec/xattrschema/db2map
```

You will see the following:

```
{axlmgr
 {{principal {{query r} {update m} {test r} {delete m}}}
  {group     {{query r} {update m} {test r} {delete m}}}}}
{annotation {Schema entry for DB2 database access}}
{applydefs no}
{intercell rejects}
{multivalued no}
{reseved no}
{scope {}}
{trigbind {}}
{trigype none}
{unique no}
{uuid 1cbe84ca-9df3-11cf-84cd-02608c2cd17b}
```

Note: Restrictions on the contents of the authorization name recorded in the ERA are not enforced by DCE. If a DCE principal or group is given an invalid authorization name, an error results when an attempt is made by DB2 to authenticate that user. (Recall that authentication may occur at CONNECT, ATTACH, DB2START, or any other operation where authentication is required.) It is also highly recommended that you ensure the assignment of authorization names to DCE principals is one-to-one and unique. DCE does not check these conditions.

If a DB2 client is to access a DB2 UDB server, once they are registered as DCE principals, the ERA information must be added to provide the mapping from the principal name to the authorization name. This must be done once for each user or group; and, is accomplished by completing the following steps:

- Login to DCE as a valid DCE administrator
- Invoke `dcecp` and at the prompt enter the following:

  ```
  > principal modify principal_name \
  > -add {db2map map_1 map_2...map_n}
  ```

  where `map_n` uses the following format:

  `DCE_server_principal,DB2_authid`

where DCE_server_principal is a valid DCE principal name for a DB2 UDB server (or is the wildcard * which indicates this mapping is valid for any DB2 server not already specified in another map_n entry) and DB2_authid is a valid DB2 authorization name.

How to Setup a DB2 Server to Use DCE

Servers must be registered in the Distributed Computing Environment (DCE) Registry and have correct attributes before being used with DB2. See the appropriate platform-specific DCE documentation for information on how to create a DCE server principal.

The DCE Security client runtime code must be installed and accessible by the server instance.

Each DB2 server that wishes to use DCE as an authentication mechanism must register with DCE at the time of issuing DB2START. To avoid having to do this manually, DCE provides a method whereby a server maintains its own user ID and password (key) information in a special file called a *keytab* file. At DB2START, DB2 reads the database manager configuration file and obtains the authentication type for the instance. If it finds the authentication type is DCE, DCE calls are made by the DB2 server to obtain the information from the keytab file. It is this information that is used to register the server with DCE. This registration allows the server to accept DCE tokens from DCE clients and to use them to authenticate these users.

The instance administrator must create the keytab file for the instance using DCE commands. Detailed information on how to create a keytab file is included in the DCE documentation for your platform. In that document, refer to the details associated with the keytab file and the commands *dcecp keytab* or *rgy_edit*. The DB2 keytab file must be named *keytab.db2* and must reside in the security subdirectory of the sqllib directory for the instance. (For Intel-based operating systems, the file must reside in the security subdirectory of the INSTANCENAME subdirectory of the sqllib directory. INSTANCENAME is the instance name of the database you are working with.) It should contain only one entry for the server principal for the specified instance; anything else results in an error at DB2START time. On UNIX operating system platforms, this file must be protected with file permissions to only allow read/write for the instance owner.

Following is an example of the creation of the keytab file:

- Login to DCE as a valid DCE user
- Invoke rgy_edit, and enter the following at the prompt:

```
> ktadd -p principal_name -pw principal_password \
> -f keytab.db2
```

To start DB2 using DCE authentication once the DCE configuration is complete, you must tell DB2 it is to use DCE authentication by updating the database manager configuration file with authentication type “DCE.” This is done by issuing the following CLP command:

```
update database manager configuration using authentication DCE
  sysadm_group DCE_group_name
```

Then perform a `dce_login` to a valid DB2 DCE user and issue DB2START.

Note: Before starting DB2 using DCE authentication, ensure you have defined a DCE user principal to be used as your SYSADM for the instance so that you have a valid DCE user ID from which to start, stop, and administer the instance. Please see "How to Setup a DB2 User for DCE" on page 117 for instructions on how to do this.

In addition to these instructions, ensure the principal created is a member of the SYSADM_GROUP for the instance. By default, this group name is DB2ADMIN for DCE authentication when no group is explicitly specified (that is, when the SYSADM_GROUP is null), but it can be updated before changing the authentication type for the instance to a group name (authorization name) of your choice. The DCE group that you select must have an ERA defined that maps it to the specified SYSADM_GROUP authorization name.

How to Setup a DB2 Client Instance to Use DCE

A client-only instance may be established to use DCE authentication for local operations by updating the database manager configuration file and setting the authentication type to DCE. There is no requirement to have a keytab file for a client-only instance since there is no server that needs to register to DCE. In general, it is not recommended (or required) that a client-only DB2 instance use DCE authentication, but it is supported.

A client that wishes to access a remote database using DCE security requires access to the applicable DCE Security product. Optionally, the client may choose to catalog the authentication type for the target database in the database directory. If the client chooses to specify DCE authentication, the fully-qualified DCE server principal name must also be specified. If DCE authentication is not specified in the directory, the authentication and principal information is obtained from the server at CONNECT time.

DB2 Restrictions Using DCE Security

Using DCE authentication places some restrictions on certain SQL functions provided by DB2 and related to group support. The following restrictions exist when using DCE authentication:

1. When using the GRANT or REVOKE statements, the keywords USER and GROUP **must** be specified to qualify the authorization name specified, otherwise an error is issued.
2. When using the AUTHORIZATION clause of the CREATE SCHEMA statement, the group membership of the authorization name specified will **not** be considered in evaluating the authorizations required to perform the statements that follow this clause. This may result in an authorization failure during execution of the CREATE SCHEMA statement.
3. When a package is rebound by a user other than the original binder of the package, the privileges of the original binder are reevaluated. In this case, group

membership of the original binder are not considered when reevaluating privileges. This may result in an authorization failure during rebinding.

Privileges, Authorities, and Authorization

Privileges enable users to create or access database resources. *Authority levels* provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate *authorization*, that is, the required privilege or authority.

The following authorities exist:

- “System Administration Authority (SYSADM)” on page 123
- “Database Administration Authority (DBADM)” on page 125
- “System Control Authority (SYSCTRL)” on page 124
- “System Maintenance Authority (SYSMAINT)” on page 124

The following types of privileges exist:

- “Database Privileges” on page 126
- “Schema Privileges” on page 127
- “Table and View Privileges” on page 127
- “Package Privileges” on page 128
- “Index Privileges” on page 129

Figure 19 on page 122 illustrates the hierarchical relationship between authorities and privileges. In the hierarchy, the lower level authorities and privileges are subsets of those above them.

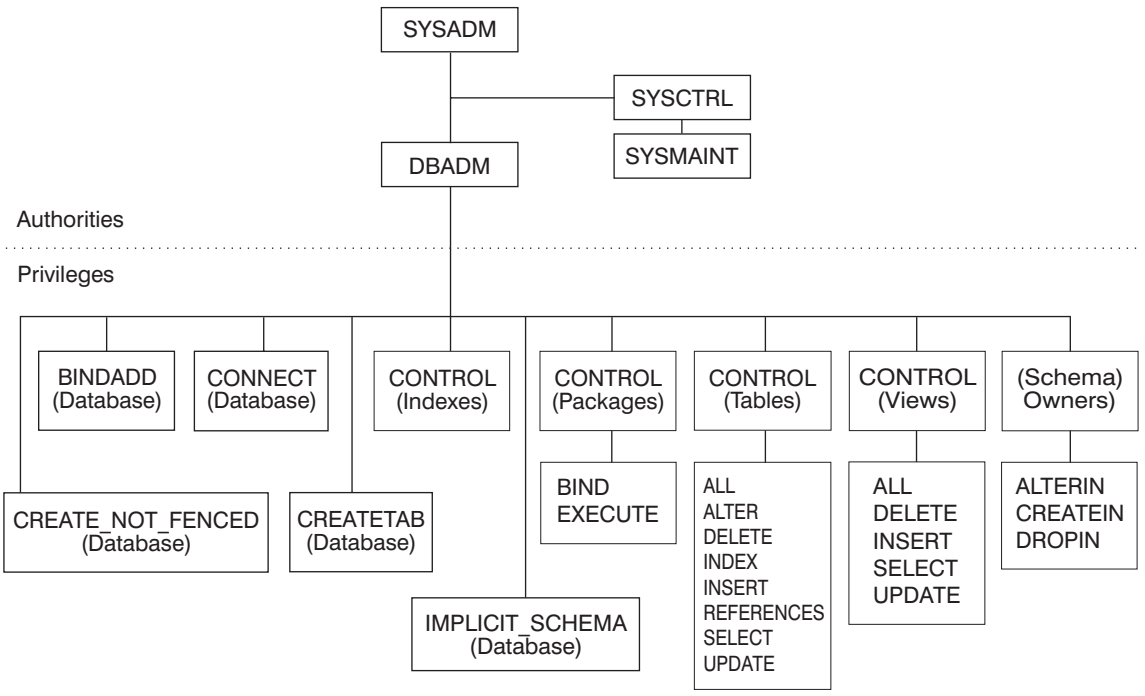


Figure 19. Hierarchy of Authorities and Privileges

A user or group can have one or more of the following levels of authorization:

- Administrative authority (SYSADM or DBADM) gives full privileges for a set of objects.
- System authority (SYSCTRL or SYSMAINT) gives full privileges for managing the system, but does not allow access to the data.
- Ownership privilege (also called CONTROL privilege in some cases) gives full privileges for a specific object.
- Individual privileges may be granted to allow a user to carry out specific functions on specific objects.
- Implicit privileges may be granted to a user who has the privilege to execute a package. While users can run the application, they do not necessarily require explicit privileges on the data objects used within the package. For more information see “Allowing Indirect Privileges through a Package” on page 132.

Users with administrative authority (SYSADM or DBADM) or ownership privileges (CONTROL) can grant and revoke privileges to and from others, using the GRANT and REVOKE statements. (See “Controlling Access to Database Objects” on page 129.) It is also possible to grant a table, view, or schema privilege to another user if that privilege is held WITH GRANT OPTION. However, the WITH GRANT OPTION does

not allow the person granting the privilege to revoke the privilege once granted. You must have SYSADM authority, DBADM authority, or CONTROL privilege to revoke the privilege.

A user or group can be authorized for any combination of individual privileges or authorities. When a privilege is associated with a resource, that resource must exist. For example, a user cannot be given the SELECT privilege on a table unless that table has previously been created.

Note: Care must be taken when an authorization name is given authorities and privileges and there is no user created with that authorization name. At some later time, a user can be created with that authorization name and automatically receive all of the authorities and privileges associated with that authorization name.

See the *Command Reference*, the *API Reference*, or the *SQL Reference* for information about what authorization is required for a particular command, API, or SQL statement.

System Administration Authority (SYSADM)

SYSADM authority is the highest level of administrative authority. Users with SYSADM authority can run utilities, issue database and database manager commands, and access the data in any table in any database within the database manager instance. It provides the ability to control all database objects in the instance, including databases, tables, views, indexes, packages, schemas, aliases, data types, functions, procedures, triggers, table spaces, nodegroups, buffer pools, and event monitors.

SYSADM authority is assigned to the group specified by the *sysadm_group* configuration parameter (see "System Administration Authority Group Name (sysadm_group)" on page 570). Membership in that group is controlled outside the database manager through the security facility used on your platform. See the *Quick Beginnings* for information on how to use your system security facility to create, change, or delete SYSADM authorities.

Only a user with SYSADM authority can perform the following functions:

- Migrate a database
- Change the database manager configuration file (including specifying the groups having SYSCTRL or SYSMAINT authority)
- Grant DBADM authority.

In addition, a user with SYSADM authority can perform the functions of users with the following authorities:

- "System Control Authority (SYSCTRL)" on page 124
- "System Maintenance Authority (SYSMAINT)" on page 124
- "Database Administration Authority (DBADM)" on page 125

Note: When users with SYSADM authority create databases, they are automatically granted explicit DBADM authority on the database. If the database creator is removed from the SYSADM group, and if you want to also prevent them from

accessing that database as a DBADM, you must explicitly revoke this DBADM authority.

System Control Authority (SYSCTRL)

SYSCTRL authority is the highest level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases. System control authority is designed for users administering a database manager instance containing sensitive data.

SYSCTRL authority is assigned to the group specified by the *sysctrl_group* configuration parameter (see "System Control Authority Group Name (sysctrl_group)" on page 571). If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSCTRL authority or higher can do the following:

- Update a database, node, or distributed connection services (DCS) directory
- Force users off the system
- Create or drop a database
- Drop, create, or alter a table space
- Restore to new database.

In addition, a user with SYSCTRL authority can perform the functions of users with "System Maintenance Authority (SYSMAINT)" authority.

Users with SYSCTRL authority also have the implicit privilege to connect to a database.

Note: When users with SYSCTRL authority create databases, they are automatically granted explicit DBADM authority on the database. If the database creator is removed from the SYSCTRL group, and if you want to also prevent them from accessing that database as a DBADM, you must explicitly revoke this DBADM authority.

System Maintenance Authority (SYSMAINT)

SYSMAINT authority is the second level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases. System maintenance authority is designed for users maintaining databases within a database manager instance that contains sensitive data.

SYSMAINT authority is assigned to the group specified by the *sysmaint_group* configuration parameter (see "System Maintenance Authority Group Name (sysmaint_group)" on page 572). If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSMAINT or higher system authority can do the following:

- Update database configuration files
- Backup a database or table space
- Restore to an existing database
- Perform roll forward recovery
- Start or stop a database instance
- Restore a table space
- Run trace
- Take database system monitor snapshots of a database manager instance or its databases.

A user with SYSMAINT, DBADM, or higher authority can do the following:

- Query the state of a table space
- Update log history files
- Quiesce a table space
- Reorganize a table
- Collect catalog statistics using the RUNSTATS utility.

Users with SYSMAINT authority also have the implicit privilege to connect to a database.

Database Administration Authority (DBADM)

DBADM authority is the second highest level of administrative authority. It applies only to a specific database, and allows the user to run certain utilities, issue database commands, and access the data in any table in the database. When DBADM authority is granted, BINDADD, CONNECT, CREATETAB, CREATE_NOT_FENCED, and IMPLICIT_SCHEMA privileges are granted as well. Only a user with SYSADM authority can grant or revoke DBADM authority. Users with DBADM authority can grant privileges on the database to others and can revoke any privilege from any user regardless of who granted it.

Only a user with DBADM or higher authority can do the following:

- Read log files
- Create, activate and drop event monitors
- Run the load utility.

A user with DBADM, SYSMAINT, or higher authority can do the following:

- Query the state of a table space
- Update log history files
- Quiesce a table space.
- Reorganize a table
- Collect catalog statistics using the RUNSTATS utility.

Note: A DBADM can only perform the above functions on the database for which DBADM authority is held.

Database Privileges

Database privileges involve actions on a database as a whole:

- CONNECT allows a user to access the database
- BINDADD allows a user to create new packages in the database
- CREATETAB allows a user to create new tables in the database
- CREATE_NOT_FENCED allows a user to create a user-defined function (UDF) or procedure that is "not fenced." UDFs or procedures that are "not fenced" must be extremely well tested because the database manager does not protect its storage or control blocks from these UDFs or procedures. (As a result, a poorly written and tested UDF or procedure that is allowed to run "not fenced" can cause serious problems for your system.) (See the *Embedded SQL Programming Guide* or the *SQL Reference* for more information.)
- IMPLICIT_SCHEMA allows any user to create a schema implicitly by creating an object using a CREATE statement with a schema name that does not already exist. SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema.

Only users with SYSADM or DBADM authority can grant and revoke these privileges to and from other users.

Note: When a database is created, the following privileges are automatically granted to PUBLIC:

- CREATETAB
- BINDADD
- CONNECT
- IMPLICIT_SCHEMA
- SELECT privilege on the system catalog views.

To remove any privilege, a DBADM or SYSADM must explicitly revoke the privilege from PUBLIC.

Implicit Schema Authority (IMPLICIT_SCHEMA) Considerations

When a new database is created, or when a database is migrated from the previous release, PUBLIC is given IMPLICIT_SCHEMA database authority. With this authority, any user can create a schema by creating an object and specifying a schema name that does not already exist. SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema.

If control of who can implicitly create schema objects is required for the database, IMPLICIT_SCHEMA database authority should be revoked from PUBLIC. Once this is done, there are only three (3) ways that a schema object is created:

- Any user can create a schema using their own authorization name on a CREATE SCHEMA statement.
- Any user with DBADM authority can explicitly create any schema which does not already exist, and can optionally specify another user as the owner of the schema.

- Any user with DBADM authority has IMPLICIT_SCHEMA database authority (independent of PUBLIC) so that they can implicitly create a schema with any name at the time they are creating other database objects. SYSIBM becomes the owner of the implicitly created schema and PUBLIC has the privilege to create objects in the schema.

A user always has the ability to explicitly create their own schema using their own authorization name.

Schema Privileges

Schema privileges involve actions on schemas in a database. A user may be granted any of the following privileges:

- CREATEIN allows the user to create objects within the schema.
- ALTERIN allows the user to alter objects within the schema.
- DROPIN allows the user to drop objects from within the schema.

The owner of the schema has all of these privileges and the ability to grant them to others. The objects that are manipulated within the schema object include: tables, views, indexes, packages, data types, functions, triggers, procedures, and aliases.

Table and View Privileges

Table and view privileges involve actions on tables or views in a database. A user must have CONNECT privilege on the database to use any of the following privileges:

- CONTROL provides the user with all privileges for a table or view including the ability to drop it, and to grant and revoke individual table privileges. You must have SYSADM or DBADM authority to grant CONTROL. The creator of a table automatically receives CONTROL privilege on the table. The creator of a view automatically receives CONTROL privilege only if they have CONTROL privilege on all tables and views referenced in the view definition, or they have SYSADM or DBADM authority.
- ALTER allows the user to add columns to a table, to add or change comments on a table and its columns, to add a primary key or unique constraint and to create or drop a table check constraint. The user can also create triggers on the table, although additional authority on all the objects referenced in the trigger (including SELECT on the table if the trigger references any of the columns of the table) is required. A user with ALTER privilege on all the descendent tables can drop a primary key; a user with ALTER privilege on the table and REFERENCES privilege on the parent table, or REFERENCES privilege on the appropriate columns, can create or drop a foreign key. A user with ALTER privilege can also COMMENT ON a table.
- DELETE allows the user to delete rows from a table or view.
- INDEX allows the user to create an index on a table. Creators of indexes automatically have CONTROL privilege on the index. For more information, see “Index Privileges” on page 129.
- INSERT allows the user to insert an entry into a table or view, and to run the IMPORT utility.

- REFERENCES allows the user to create and drop a foreign key, specifying the table as the parent in a relationship. The user may have this privilege only on specific columns.
- SELECT allows the user to retrieve rows from a table or view, to create a view on a table, and to run the EXPORT utility.
- UPDATE allows the user to change an entry in a table, a view, or for one or more specific columns in a table or view. The user may have this privilege only on specific columns.

The privilege to grant these privileges to others may also be granted using the WITH GRANT OPTION on the GRANT statement.

Note: When a user or group is granted CONTROL privilege on a table, all other privileges on that table are automatically granted WITH GRANT OPTION. If you subsequently revoke the CONTROL privilege on the table from a user, that user will still retain the other privileges that were automatically granted. To revoke all the privileges that are granted with the CONTROL privilege, you must either explicitly revoke each individual privilege or specify the ALL keyword on the REVOKE statement, for example:

```
REVOKE ALL
  ON EMPLOYEE FROM USER HERON
```

The following manuals provide information about the authorizations required to execute specific commands, APIs, or SQL statements:

- *SQL Reference*
- *Command Reference*
- *API Reference*.

See “User Update-Capable Catalog Statistics” on page 330 for information about the authorization required to update catalog statistics.

For information about how view privileges are determined, see the CREATE VIEW statement in the *SQL Reference* manual.

Package Privileges

A package is a database object that contains the information needed by the database manager to access data in the most efficient way for a particular application program. Package privileges enable a user to create and manipulate packages. The user must have CONNECT privilege on the database to use any of the following privileges:

- CONTROL provides the user with the ability to rebind, drop, or execute a package as well as the ability to extend those privileges to others. The creator of a package automatically receives this privilege. A user with CONTROL privilege is granted the BIND and EXECUTE privileges, and can grant BIND and EXECUTE privileges to other users as well. To grant CONTROL privilege, the user must have SYSADM or DBADM authority.
- BIND allows the user to rebind an existing package.
- EXECUTE allows the user to execute a package.

In addition to these package privileges, the BINDADD database privilege allows users to create new packages or rebind an existing package in the database.

Index Privileges

The creator of an index automatically receives CONTROL privilege on the index. CONTROL privilege on an index is really the ability to drop the index. To grant CONTROL privilege on an index, a user must have SYSADM or DBADM authority.

The table-level INDEX privilege allows a user to create an index on that table (see "Table and View Privileges" on page 127).

Controlling Access to Database Objects

Controlling data access requires an understanding of direct and indirect privileges, administrative authorities, and packages. This section explains these topics and provides some examples.

Directly granted privileges are stored in the system catalog. Methods for auditing the implementation of the database access control plan are discussed in "Using the System Catalog" on page 136.

Authorization is controlled in three ways:

- Explicit authorization is controlled through privileges controlled with the GRANT and REVOKE statements
- Implicit authorization is controlled by creating and dropping objects
- Indirect privileges are associated with packages.

The following topics are discussed:

Granting Privileges

The GRANT statement allows an authorized user to grant privileges. A privilege can be granted to one or more authorization names in one statement; or to PUBLIC, which makes the privileges available to all users. Note that an authorization name can be either an individual user or a group.

On operating systems where users and groups exist with the same name, you should specify whether you are granting the privilege to the user or group. Both the GRANT and REVOKE statements support the keywords USER and GROUP. If these optional keywords are not used, the database manager checks the operating system security facility to determine whether the authorization name identifies a user or a group. If the authorization name could be both a user and a group, an error is returned.

The following example grants SELECT privileges on the EMPLOYEE table to the user HERON:

```
GRANT SELECT
   ON EMPLOYEE TO USER HERON
```

The following example grants SELECT privileges on the EMPLOYEE table to the group HERON:

```
GRANT SELECT
   ON EMPLOYEE TO GROUP HERON
```

To grant privileges on most database objects, the user must have SYSADM authority, DBADM authority, or CONTROL privilege on that object; or, the user must hold the privilege WITH GRANT OPTION. Privileges can be granted only on existing objects. To grant CONTROL privilege to someone else, the user must have SYSADM or DBADM authority. To grant DBADM authority, the user must have SYSADM authority.

See the *SQL Reference* for more information about the GRANT statement.

Revoking Privileges

The REVOKE statement allows authorized users to revoke privileges previously granted to other users. To revoke privileges on database objects, you must have DBADM authority, SYSADM authority, or CONTROL privilege on that object. Note that holding a privilege WITH GRANT OPTION is not sufficient to revoke that privilege. To revoke CONTROL privilege from another user, you must have SYSADM or DBADM authority. To revoke DBADM authority, you must have SYSADM authority. Privileges can only be revoked on existing objects.

Note: A user without DBADM authority or CONTROL privilege on a table or view is not able to revoke a privilege that they granted through their use of the WITH GRANT OPTION. Also, there is no cascade on the revoke to those who have received privileges granted by the person being revoked. For more information on the authority required to revoke privileges, see the *SQL Reference* manual.

If a privilege has been granted to both a user and a group with the same name, you must specify the GROUP or USER keyword when revoking the privilege. The following example revokes the SELECT privilege on the EMPLOYEE table from the user HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM USER HERON
```

The following example revokes the SELECT privilege on the EMPLOYEE table from the group HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM GROUP HERON
```

Note that revoking a privilege from a group may not revoke it from all members of that group. If an individual name has been directly granted a privilege, it will keep it until that privilege is directly revoked.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

If an explicitly-granted table (or view) privilege is revoked from a user with DBADM authority, privileges **will not** be revoked from other views defined on that table. This is because the view privileges are available through the DBADM authority and are not dependent on explicit privileges on the underlying tables.

If you have defined a view based on one or more underlying tables or views and you lose the SELECT privilege to one or more of those tables or views, then the view cannot be used.

Note: When CONTROL privilege is revoked from a user on a table or a view, the user continues to have the ability to grant privileges to others. When given CONTROL privilege, the user also receives all other privileges WITH GRANT OPTION. Once CONTROL is revoked, all of the other privileges remain WITH GRANT OPTION until they are explicitly revoked.

All packages that are dependent on revoked privileges are marked invalid, but can be validated if rebound by a user with appropriate authority. Packages can also be rebuilt if the privileges are subsequently granted again to the binder of the application; running the application will trigger a successful implicit rebind. If privileges are revoked from PUBLIC, all packages bound by users having only been able to bind based on PUBLIC privileges are invalidated. If DBADM authority is revoked from a user, all packages bound by that user are invalidated including those associated with database utilities. Attempting to use a package that has been marked invalid causes the system to attempt to rebind the package. If this rebind attempt fails, an error occurs (SQLCODE -727). In this case, the packages must be explicitly rebound by a user with:

- Authority to rebind the packages
- Appropriate authority for the objects used within the packages

These packages should be rebound at the time the privileges are revoked. See the *SQL Reference* for more information about the REVOKE and REBIND PACKAGE statements.

If you have defined a trigger based on one or more privileges and you lose one or more of those privileges, then the trigger cannot be used.

Managing Implicit Authorizations by Creating and Dropping Objects

The database manager implicitly grants certain privileges to a user who issues a CREATE SCHEMA, CREATE TABLE, CREATE VIEW, or CREATE INDEX statement, or who creates a new package using a PREP or BIND command. Privileges are also granted when objects are created by users with SYSADM or DBADM authority. Similarly, privileges are removed when an object is dropped.

When the created object is a table, index, or package, the user receives CONTROL privilege on the object. When the object is a view, the CONTROL privilege for the view

is granted implicitly only if the user has CONTROL privilege for all tables and views referenced in the view definition.

When the object explicitly created is a schema, the schema owner is given ALTERIN, CREATEIN, and DROPIN privileges WITH GRANT OPTION. An implicitly created schema has CREATEIN granted to PUBLIC.

For information about how view privileges are determined, see the CREATE VIEW statement in the *SQL Reference* manual.

Allowing Indirect Privileges through a Package

Access to data within a database can be requested by application programs, as well as by persons engaged in an interactive workstation session. A package contains statements that allow users to perform a variety of actions on many database objects. Each of these actions requires one or more privileges.

Privileges granted to individuals binding the package and to PUBLIC are used for authorization checking when static SQL is bound. Privileges granted through groups are **not** used for authorization checking when static SQL is bound. The user who binds a package must either have been explicitly granted all the privileges required to execute the static SQL statements in the package or have been implicitly granted the necessary privileges through PUBLIC. PUBLIC, group, and user privileges **are all** used when checking to ensure the user has the appropriate authorization (BIND or BINDADD privilege) to bind the package.

Packages may include both static and dynamic SQL. To process a package with static SQL, a user need only have EXECUTE privilege on the package. This user can then indirectly obtain the privileges of the package binder for any static SQL in the package but only within the restrictions imposed by the package.

To process a package with any dynamic SQL statements, the user must have EXECUTE privilege on the package. The user needs EXECUTE privilege on the package plus any privileges required to execute the dynamic SQL statements in the package. The binder's authorities and privileges are used for any static SQL in the package.

Controlling Access to Data with Views

A view provides a means of controlling access or extending privileges to a table by allowing:

- Access only to designated columns of the table.

  For users and application programs that require access only to specific columns of a table, an authorized user can create a view to limit the columns addressed only to those required.

- Access only to a subset of the rows of the table.

  By specifying a WHERE clause in the subquery of a view definition, an authorized user can limit the rows addressed through a view.

To create a view, a user must have SYSADM authority, DBADM authority, or CONTROL or SELECT privilege for each table or view referenced in the view definition. The user must also be able to create an object in the schema specified for the view. That is, CREATEIN privilege for an existing schema or IMPLICIT_SCHEMA authority on the database if the schema does not already exist. See “Creating a View” on page 93 for more information.

The following scenario illustrates how views can restrict access to information.

Many people may require access to information in the STAFF table, for different reasons. For example:

- The personnel department needs to be able to update and look at the entire table.

  This requirement can be easily met by granting SELECT and UPDATE privileges on the STAFF table to the group PERSONNL:

  ```
  GRANT SELECT,UPDATE ON TABLE STAFF TO GROUP PERSONNL
  ```

- Individual department managers need to look at the salary information for their employees.

  This requirement can be met by creating a view for each department manager. For example, the following view can be created for the manager of department number 51:

  ```
  CREATE VIEW EMP051 AS
     SELECT NAME,SALARY,JOB FROM STAFF
     WHERE DEPT=51
  GRANT SELECT ON TABLE EMP051 TO JANE
  ```

  The manager with the authorization name JANE would query the EMP051 view just like the STAFF table. When accessing the EMP051 view of the STAFF table, this manager views the following information:

NAME	SALARY	JOB
Fraye	45150.0	Mgr
Williams	37156.5	Sales
Smith	35654.5	Sales
Lundquist	26369.8	Clerk
Wheeler	22460.0	Clerk

- All users need to be able to locate other employees. This requirement can be met by creating a view on the NAME column of the STAFF table and the LOCATION column of the ORG table, and by joining the two tables on their respective DEPT and DEPTNUMB columns:

  ```
  CREATE VIEW EMPLOCS AS
     SELECT NAME, LOCATION FROM STAFF, ORG
     WHERE STAFF.DEPT=ORG.DEPTNUMB
  GRANT SELECT ON TABLE EMPLOCS TO PUBLIC
  ```

  Users who access the employee location view will see the following information:

NAME	LOCATION
Molinare	New York
Lu	New York
Daniels	New York
Jones	New York
Hanes	Boston
Rothman	Boston
Ngan	Boston
Kermisch	Boston
Sanders	Washington
Pernal	Washington
James	Washington
Sneider	Washington
Marenghi	Atlanta
O'Brien	Atlanta
Quigley	Atlanta
Naughton	Atlanta
Abrahams	Atlanta
Koonitz	Chicago
Plotz	Chicago
Yamaguchi	Chicago
Scoutten	Chicago
Fraye	Dallas
Williams	Dallas
Smith	Dallas
Lundquist	Dallas
Wheeler	Dallas
Lea	San Francisco
Wilson	San Francisco
Graham	San Francisco
Gonzales	San Francisco
Burke	San Francisco
Quill	Denver
Davis	Denver
Edwards	Denver
Gafney	Denver

Tasks and Required Authorizations

Not all organizations divide job responsibilities in the same manner. Table 21 lists some other common job titles, the tasks that usually accompany them, and the authorities or privileges that are needed to carry out those tasks.

Table 21. Common Job Titles, Tasks, and Required Authorization

JOB TITLE	TASKS	REQUIRED AUTHORIZATION
Department Administrator	Oversees the departmental system; creates databases	SYSCTRL authority. SYSADM authority if the department has its own instance.
Security Administrator	Authorizes other users for some or all authorizations and privileges	SYSADM or DBADM authority.
Database Administrator	Designs, develops, operates, safeguards, and maintains one or more databases	DBADM and SYSMAINT authority over one or more databases. SYSCTRL authority in some cases.
System Operator	Monitors the database and carries out backup functions	SYSMAINT authority.
Application Programmer	Develops and tests the database manager application programs; may also create tables of test data	BINDADD, BIND on an existing package, CONNECT and CREATETAB on one or more databases, some specific schema privileges, and a list of privileges on some tables.
User Analyst	Defines the data requirements for an application program by examining the system catalog views	SELECT on the catalog views; CONNECT on one or more databases.
Program End User	Executes an application program	EXECUTE on the package; CONNECT on one or more databases. See the note following this table.
Information Center Consultant	Defines the data requirements for a query user; provides the data by creating tables and views and by granting access to database objects	DBADM authority over one or more databases.
Query User	Issues SQL statements to retrieve, add, delete, or change data; may save results as tables	CONNECT on one or more databases; CREATEIN on the schema of the tables and views being created; and, SELECT, INSERT, UPDATE, DELETE on some tables and views.

If an application program contains dynamic SQL statements, the Program End User may need other privileges in addition to EXECUTE and CONNECT (such as SELECT, INSERT, DELETE, and UPDATE).

Using the System Catalog

Information about each database is automatically maintained in a set of views called the system catalog, which is created when the database is generated. This system catalog describes tables, columns, indexes, programs, privileges, and other objects.

Six of these views list the privileges held by users and the identity of the user granting each privilege:

SYSCAT.DBAUTH	Lists the database privileges
SYSCAT.TABAUTH	Lists the table and view privileges
SYSCAT.COLAUTH	Lists the column privileges
SYSCAT.PACKAGEAUTH	Lists the package privileges
SYSCAT.INDEXAUTH	Lists the index privileges
SYSCAT.SCHEMAAUTH	Lists the schema privileges

Privileges granted to users by the system will have SYSIBM as the grantor. SYSADM, SYSMAINT and SYSCTRL are not listed in the system catalog.

The CREATE and GRANT statements place privileges in the system catalog. Users with SYSADM and DBADM authorities can grant and revoke SELECT privilege on the system catalog views. The following examples show how to extract information about privileges by using these SQL queries:

- "Retrieving Authorization Names with Granted Privileges"
- "Retrieving All Names with DBADM Authority" on page 137
- "Retrieving Names Authorized to Access a Table" on page 137
- "Retrieving All Privileges Granted to Users" on page 137
- "Securing the System Catalog Views" on page 138

Retrieving Authorization Names with Granted Privileges

No single system catalog view contains information about all privileges. The following statement retrieves all authorization names with privileges:

```
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'DATABASE' FROM SYSCAT.DBAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'TABLE   ' FROM SYSCAT.TABAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'PACKAGE ' FROM SYSCAT.PACKAGEAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'INDEX   ' FROM SYSCAT.INDEXAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'COLUMN  ' FROM SYSCAT.COLAUTH
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SCHEMA  ' FROM SYSCAT.SCHEMAAUTH
ORDER BY GRANTEE, GRANTEETYPE, 3
```

Periodically, the list retrieved by this statement should be compared with lists of user and group names defined in the system security facility. You can then identify those authorization names that are no longer valid.

Note: If you are supporting remote database clients, it is possible that the authorization name is defined at the remote client only and not on your database server machine.

Retrieving All Names with DBADM Authority

The following statement retrieves all authorization names that have been directly granted DBADM authority:

```
SELECT DISTINCT GRANTEE FROM SYSCAT.DBAUTH
   WHERE DBADMAUTH = 'Y'
```

Retrieving Names Authorized to Access a Table

The following statement retrieves all authorization names that are directly authorized to access the table EMPLOYEE with the qualifier JAMES:

```
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
   WHERE TABNAME = 'EMPLOYEE'
     AND TABSCHEMA = 'JAMES'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
   WHERE TABNAME = 'EMPLOYEE'
     AND TABSCHEMA = 'JAMES'
```

To find out who can update the table EMPLOYEE with the qualifier JAMES, issue the following statement:

```
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
   WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
      (CONTROLAUTH  = 'Y' OR
       UPDATEAUTH   = 'Y' OR  UPDATEAUTH   = 'G')
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.DBAUTH
   WHERE DBADMAUTH = 'Y'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
   WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
   PRIVTYPE = 'U'
```

This retrieves any authorization names with DBADM authority, as well as those names to which CONTROL or UPDATE privileges have been directly granted. However, it will not return the authorization names of users who only hold SYSADM authority.

Remember that some of the authorization names may be groups, not just individual users.

Retrieving All Privileges Granted to Users

By making queries on the system catalog views, users can retrieve a list of the privileges they hold and a list of the privileges they have granted to other users. For example, the following statement retrieves a list of the database privileges that have been directly granted to an individual authorization name:

```
SELECT * FROM SYSCAT.DBAUTH
   WHERE GRANTEE = USER AND GRANTEETYPE = 'U'
```

The following statement retrieves a list of the table privileges that were directly granted by a specific user:

```
SELECT * FROM SYSCAT.TABAUTH
   WHERE GRANTOR  = USER
```

The following statement retrieves a list of the individual column privileges that were directly granted by a specific user:

```
SELECT * FROM SYSCAT.COLAUTH
   WHERE GRANTOR  = USER
```

The keyword USER in these statements is always equal to the value of a user's authorization name. USER is a read-only special register. See the *SQL Reference* for more information on special registers.

Securing the System Catalog Views

During database creation, SELECT privilege on the system catalog views is granted to PUBLIC. (See “Database Privileges” on page 126 for other privileges that are automatically granted to PUBLIC.) In most cases, this does not present any security problems. For very sensitive data, however, it may be inappropriate, as these tables describe every object in the database. If this is the case, consider revoking the SELECT privilege from PUBLIC; then grant the SELECT privilege as required to specific users. Granting and revoking SELECT on the system catalog views is done in the same way as for any view, but you must have either SYSADM or DBADM authority to do this.

At a minimum, you should consider restricting access to the SYSCAT.DBAUTH, SYSCAT.TABAUTH, SYSCAT.PACKAGEAUTH, SYSCAT.INDEXAUTH, SYSCAT.COLAUTH, and SYSCAT.SCHEMAAUTH catalog views. This would prevent information on user privileges, which could be used to target an authorization name for break-in, becoming available to everyone with access to the database.

You should also examine the columns for which statistics are gathered (see Chapter 11, “System Catalog Statistics” on page 311). Some of the statistics recorded in the system catalog contain data values which could be sensitive information in your environment. If these statistics contain sensitive data, you may wish to revoke SELECT privilege from PUBLIC for the SYSCAT.COLUMNS and SYSCAT.COLDIST catalog views.

If you wish to limit access to the system catalog views, you could define views to let each authorization name retrieve information about its own privileges.

For example, the following view MYSELECTS includes the owner and name of every table on which a user's authorization name has been directly granted SELECT privilege:

```
CREATE VIEW MYSELECTS AS
   SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABAUTH
   WHERE GRANTEETYPE = 'U'
     AND GRANTEE = USER
     AND SELECTAUTH = 'Y'
```

The keyword USER in this statement is always equal to the value of the authorization name.

The following statement makes the view available to every authorization name:

```
GRANT SELECT ON TABLE MYSELECTS TO PUBLIC
```

And finally, remember to revoke SELECT privilege on the base table:

```
REVOKE SELECT ON TABLE SYSCAT.TABAUTH FROM PUBLIC
```

Chapter 5. Utilities for Moving Data

The LOAD utility moves data into tables, extends existing indexes, and generates statistics. LOAD moves the data much faster than the IMPORT utility when large amounts of data are involved. Data, unloaded using the EXPORT utility, can be loaded with the LOAD utility.

The AutoLoader utility splits large amounts of data and loads the split data into the different partitions of a partitioned database.

The IMPORT and EXPORT utilities move data between a table or view and another database or spreadsheet program; between DB2 databases; and between DB2 databases and host databases using DB2 Connect.

DataPropagator Relational (DPROPR) is a component of DB2 Universal Database that allows automatic copying of table updates to other tables in other DB2 relational databases.

Note: Other vendor's products that move data in and out of databases are also available, but are not discussed here.

The following topics are discussed:

- Using the LOAD Utility
- Using the AutoLoader Utility
- Using the IMPORT Utility
- Using the EXPORT Utility
- LOAD, IMPORT, and EXPORT File Formats
- Moving Data Between Systems

Notes:

1. Remember to ensure you have the required file permissions when accessing data from local area networks (LANs).
2. If DB2 for Windows NT has been defined as a Service to the Windows NT operating system, the Service must have a User Account with the required Read/Write file permissions to use LAN resources (drives, directories, and files).

Using the LOAD Utility

The LOAD utility is intended for the initial load or an append of a table where large amounts of data are moved. There are no restrictions on the data types used by the LOAD utility including large objects (LOBs) and user-defined types (UDTs). The LOAD utility speeds up the task of loading large amounts of data into a database. LOAD is faster than IMPORT because LOAD writes formatted pages directly into the database while IMPORT does SQL INSERTs. The data being loaded must be local to the server (unlike IMPORT and EXPORT where data can be passed from the client).

In addition to the overview to the LOAD utility above, here are some details about the LOAD utility that may be of interest to you. The LOAD utility also almost completely eliminates the logging associated with the loading of data. In place of logging, you have the option of making a copy of the loaded portion of the table. LOAD does not fire triggers; and does not perform referential, and table, constraint checking (other than validating the uniqueness of the unique indexes). Tables with such options defined may be populated faster, or more simply, using IMPORT. If you have a recoverable database, you can do one of the following:

- Use the non-recoverable LOAD option (and not require a backup)
- Explicitly request a copy of the loaded portion of the table be made
- Take a backup of the table space(s) in which the table resides after the completion of the load.

The LOAD utility can take advantage of a hardware configuration where multiple processors and/or multiple storage devices are used such as in a symmetric multiprocessor (SMP) environment. There are several ways in which parallel processing of large amounts of data can take place using the LOAD utility. One way is through the use of multiple storage devices which allows for I/O parallelism during the LOAD process. Another way involves the use of the multiple processors in an SMP environment which allows for intra-partition parallelism. And both can be used together to provide even faster loading of the data.

The following topics provide more information:

- Overview of the LOAD Process
- Details About LOAD
- LOAD Performance Considerations
- LOAD Temporary Space Limitations
- Restarting LOAD and Database Recovery
- LOAD Exception Table
- Checking For Constraint Violations

Overview of the LOAD Process

There are multiple phases to the LOAD process: Load, where the data is written into the table; Build, where the indexes are created; and Delete, where the rows that caused a unique key violation are removed from the table. You must run the SET CONSTRAINTS SQL statement after the load completes if there are tables left in the **check pending** state to validate the table for referential integrity and check constraints. The LOAD utility generates messages about the progress of each phase. If a failure occurs during the LOAD process, these messages will assist you in deciding how to recover.

During the Load phase, data is loaded into the table; index keys and table statistics are collected if necessary. Save points, or points of consistency, are established at intervals specified by you in the `SAVECOUNT` parameter on the LOAD command. These points of consistency are not established exactly on the number of rows specified with the `SAVECOUNT` parameter; rather the number of rows are converted to a page count, and rounded up to intervals of the extent size. Messages let you know how many input rows

were successfully loaded at the time of the save point. If a failure occurs, you should use the number of input rows at the last successful consistency point with the RESTARTCOUNT parameter during a restart. If the failure occurs near the beginning of the LOAD process and you were doing a REPLACE, you might consider restarting the load using the REPLACE option.

During the Build phase, indexes are created based on the index keys collected in the Load phase. The index keys are sorted during the Load phase and index statistics are collected. The statistics collected are similar to those collected during RUNSTATS. If a failure occurs, the Build phase restarts from the beginning.

Unique key violations are placed into the exception table, if one was specified, and messages on rejected rows are put into the message file. Following the completion of the LOAD process: review these messages, correct any problems, and insert the corrected rows into the table.

Note: The recording of warnings has a detrimental effect on the performance of the LOAD. If performance is important, and you anticipate a large number of warnings, you should consider using the NOROWWARNINGS filetype modifier. If this filetype modifier was specified, these warnings are suppressed.

During the Delete phase, all rows causing a unique key violation are deleted. If a failure occurs, this phase should be restarted by you from the beginning. Information on the rows containing the invalid keys is stored in a temporary file. After you request a restart to begin at the Delete phase, the violating rows are deleted based on the information in a temporary file. You must not modify any data in any temporary files. Also, you must restart the LOAD command with the same parameters, otherwise the Delete phase will fail. If the temporary file has been modified, or does not exist, you should restart the LOAD command at the Build phase. Once the index is re-built, any invalid keys are placed in the exception table if it exists, and duplicate keys are deleted.

Since regular logging is not performed, LOAD uses **pending** states to preserve consistency of the database. The Load and Build phases of the LOAD process place any associated table spaces into a **load pending** state. The Delete phase of the LOAD process places any associated table spaces into a **delete pending** state. If you complete the LOAD process but you do not have either logretain or userexit "on"; and, you have not specified the COPY YES option nor the NONRECOVERABLE option, then any associated table spaces are placed in a **backup pending** state. These states can be checked by using the LIST TABLESPACES command. (For more information on LIST TABLESPACES, see the *Command Reference* manual.) One last possible state associated with the LOAD process is concerned with referential and check constraints. Dependent tables may be placed in a **check pending** state following the completion of the LOAD process.

LOAD can also be used with a non-recoverable option. This allows you to perform a non-recoverable LOAD without affecting the recoverability of all other tables in the database. When this type of LOAD is run, there is no requirement for either using the COPY YES option or having a backup taken.

If a LOAD fails, the table space(s) involved could be in an inconsistent state because there is no logging. For this reason, the table spaces are left in a **load pending** state. To remove the load pending state, you will have to restart the LOAD, perform a `LOAD REPLACE` on the same table on which the LOAD failed, or recover the table space(s) using a RESTORE with the most recent backup (either table space or database backup) and then carry out further recovery actions. (You could also drop the table space and then re-create it.)

For more information on how to recover, see Chapter 6, " Recovering a Database" on page 179.

Details About LOAD

The LOAD utility inserts data into a table from an input file, from a device, or using a named pipe, any of which must reside on the node where the database resides. The table must exist. (Indexes on the table may or may not exist. LOAD only builds indexes that are already defined on the table.) If the table receiving the new data already contains data, you can replace or append to the existing data.

Note: If you are loading a table that already contains data and the database is non-recoverable, make sure that you have a backed-up copy of the database, or the table space for the table being loaded, before using LOAD so that you can recover from errors.

If the existing table is a parent table containing a primary or unique key referenced by a foreign key in a dependent table, replacing data in the parent table places the dependent table in a **check pending** state. The SET CONSTRAINTS statement must then be used to validate the referential and check constraints.

The table spaces in which the loaded table resides are quiesced in exclusive mode. For more information on QUIESCE, see the *Command Reference* manual.

Note: In the following command line processor example, and in the other in the other examples in this chapter, relative path names are used. Please be aware that relative path names are only allowed on calls from a client on the same node as the database. The use of fully-qualified path names is recommended.

The following is an example of the command line processor syntax for the LOAD command:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   insert into userid.staff copy yes use adsm data buffer 20
```

This example assumes no indexes are involved, any warning or error messages are placed in `staff.msgs`, a copy of the changes made is stored in ADSTAR Distributed Storage Manager (ADSM), and 20 pages of buffer space are to be used during the load. See "ADSTAR Distributed Storage Manager" on page 230 for more information on using ADSM.

If indexes are involved, the following is an example of the command line processor syntax for the LOAD command:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   remote file stafftmp replace into staff using . sort buffer 200
```

This example is similar to the previous one except that: `stafftmp` is used as a base file name for temporary files such as “stafftmp.msg,” “stafftmp.log,” and “stafftmp.rid”; the current directory of the user is used as the working directory; and 200 pages of buffer space are used to sort index keys. For more information on the LOAD command, see the *Command Reference* manual.

Since you will typically be loading large amounts of data using the LOAD command, a LOAD QUERY command can be used to check the progress of the LOAD process if the REMOTE FILE option has been specified. You require a connection to the same database and a separate CLP session to enter this command. This command can be used by local and remote users. For more information on the LOAD QUERY command, see the *Command Reference* manual.

The LOAD utility can also be invoked by the application programming interfaces (APIs) **sqluload** and **sqlgload**. For more information on the requirements when loading data to a table using these APIs, see the *API Reference* manual.

You should also review the following points. Each represents a task that you may need to perform and each is carried out as part of the LOAD command. For more information on any of the following tasks, see the *API Reference* and/or the *Command Reference* manuals.

- Definition of the path(s) and the input filename(s) in which the LOBs are stored. The use of `LOBSINFILE` in the `MODIFIED BY` parameter will tell the LOAD utility that all LOB data is being loaded from files.
- Determine if column values being loaded have implied decimal points or not. The use of `IMPLIEDDECIMAL` in the `MODIFIED BY` parameter will tell the LOAD utility that there are decimal points to be applied to the data as it enters the table. For example, with the IMPLIEDDECIMAL, the value 12345 is loaded into a DECIMAL(8,2) column as 123.45, **NOT** 12345.00.
- Determine whether statistics will be gathered during the load process. (This option is not supported if the load is in INSERT or RESTART mode.) Gathering statistics is only valid when indexes are already defined on the table prior to the load.
- Define the method to use for loading the data: column location, column name, or relative column position.
- Determine whether to keep a copy of the changes made. (This option is not supported if forward log recovery is disabled for the database; that is, LOGRETAIN=OFF and USEREXIT=OFF.) If no copy is made, and forward log recovery is not disabled, the table space is left in a **backup pending** state at the completion of the load.

  For more information on how to recover, see Chapter 6, “ Recovering a Database” on page 179.
- Declare the location of the directories holding temporary files during the creation of indexes. The directories are defined by the `USING` parameter. When defining the directory, if it is unqualified and a local connection, the directory will be qualified using the current directory of the user running the load. If the directory is

unqualified and part of a remote connection, the load will fail. If the parameter is not defined by the `USING` parameter, the default is to create the files in the `tmp` subdirectory in the directory defined by the DB2INSTPROF environment variable. You should be aware that the amount of information stored in this directory will be at least equal to the size of all the indexes and possibly twice that size.

- For all other temporary files either a base name defined by `REMOTE FILE` is used or, the default name `db2utmp` is used. There are many temporary files that the load process may create: The temporary files for sorting go in the path specified by the USING option; other temporary files go in the path specified by the REMOTE FILE option. Only the remote message temporary file is of any interest to you. You can use the LOAD QUERY command to save the messages generated by the load process to a file that can later be viewed as output. See the *Command Reference* for further information on these topics.

  The remote file resides on the server machine and is accessed by the DB2 instance exclusively. Therefore, any filename qualification given to this parameter must reflect the directory structure of the server, not the client, and the DB2 instance owner must have read and write permission on the file. In addition, the user must ensure that two loads are not issued having the same fully-qualified remote filename.

  There are several ways in which the remote filename may be selected and qualified when the user has just given a partially qualified name, or no name at all.

  Notes:

  1. No remote filename is given in a load operation where the user is on the **same** machine as the database instance. In this case, the load utility uses the name `db2utmp` and qualifies it with the current working directory of the user. Two loads from the same directory with this option will clash in the use of the remote filename. Therefore, this option is not recommended.
  2. No remote filename is given in a load operation where the user is on a **different** machine than the database instance. In this case, the load utility will generate a name that will reside in the database directory. This effectively prevents the user from using the load query facility, since the facility requires the name of the remote file. In addition, the filename generated is not guaranteed to be unique, and therefore clashes may occur between different load operations. Therefore, this option is not recommended.
  3. A non-fully-qualified filename is given in a load operation, where the user is on the **same** machine as the database instance. In this case, the name is qualified by using the current directory of the user. The user must ensure that the two loads are not issued from the same directory with the same remote filename.
  4. A non-fully-qualified filename is given in a load operation, where the user is on a **different** machine than the database instance. In this case, the load utility will reject the filename. It must be fully-qualified from the client.
  5. A fully-qualified filename is given in a load operation. This will be the filename used. The user must ensure that two loads are not issued with the same remote filename. This is the recommended usage.

You should use a remote filename if you wish to use the LOAD QUERY tool. It is difficult to determine the remote filename if you do not provide the one to be used. For more information on LOAD and the importance of the filename, see “Restarting the LOAD” on page 150.

To load data into a table, you must have either SYSADM or DBADM authority.

There are restrictions and limitations with the LOAD utility:

- Attempts to create or drop tables in a table space that is in **load pending** state will fail.
- You cannot load data into a database accessed through DB2 Connect or to a downlevel server other than DB2 Version 2.

  Note: Options only available with this release of DB2 cannot be used with a server from the previous release.

- If a `REPLACE` is performed on a table and an error occurs during the LOAD, the original data in the table is lost. Keep the LOAD input to allow the LOAD to be restarted if an error occurs during the process.
- If there is no value for a NOT NULL column, the row is rejected since the LOAD utility attempts to load a NULL value. This condition also applies when there is no value for a `NOT NULL WITH DEFAULT` column. If a `NOT NULL WITH DEFAULT` column is not one of the columns being loaded, it is filled with the `DEFAULT` values.
- Triggers are not activated on newly-loaded rows. Any business rules forming these triggers are **not** enforced by the LOAD command.

The LOAD utility optionally updates table and index statistics as part of the load process if run in REPLACE mode. If data is appended into a table, statistics **are not** gathered for the table. Run RUNSTATS following the completion of the load process to collect up-to-date statistics for the table. If gathering statistics on a table with a unique index, and duplicate keys are deleted during the Delete phase, then the statistics are not updated to account for the deleted records. If you think you will have duplicate records, you should not collect statistics during the LOAD but run RUNSTATS after the LOAD process.

To ensure the loaded data doesn't cause referential integrity or check constraint violations, any loaded table that is a parent table will cause all dependent tables to be placed in a **check pending** state. To remove the table from the restricted state, you must run the SET CONSTRAINTS statement.

With LOAD, there is a possibility of unequal code page situations involving possible expansion or contraction of the character data. Such situations could occur with Japanese or Traditional-Chinese Extended UNIX Code (EUC) and double-byte character sets (DBCS) which may have different length encodings for the same character. An option, NOCHECKLENGTHS, is used to toggle between two situations:

1. Comparison of input data length to target column length is performed before reading in any data. If the input length is larger than the target, NULLs are inserted for that column if it is nullable. Otherwise, the request is rejected. This is the default.

2. No initial comparison is performed and, on a row-by-row basis, an attempt is made to load the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is loaded. Specifying `NOCHECKLENGTHS` will enable this behavior.

LOAD Performance Considerations

The performance of LOAD depends on the nature and size of the data, the number of indexes, the options used, and whether the SET CONSTRAINTS statement is required. Use of SET CONSTRAINTS lengthens the total time needed to load the table and make it usable again. (For more information on the SET CONSTRAINTS statement, see the *SQL Reference* manual.)

The LOAD utility performs almost equally well in either INSERT mode or REPLACE mode.

Index creation will reduce the performance of the load process, especially when data is added to a table already containing data. If there are many indexes on a table which already has a large amount of data and only a small percentage of data to be loaded, you should consider using the IMPORT utility instead of the LOAD utility. Unique indexes also reduce the performance of the load process if duplicates are encountered. **In most cases it is still more efficient to create the index during the LOAD than to complete the LOAD and then use the CREATE INDEX command for each of the indexes.**

The LOAD utility automatically attempts to provide the best performance possible by determining the best values for `DISK_PARALLELISM`, `CPU_PARALLELISM`, `DATA_BUFFER`, and `SORT_BUFFER` if these parameters have not be specified by the user at the time the utility is run. Optimization of these values is done based on the size and the free space available in the utility heap. Consider allowing the LOAD utility to choose the values for these parameters and then attempt to tune the parameters for your particular needs.

Performance of the LOAD can be improved by:

- Using the `FASTPARSE` option in the `MODIFIED BY` parameter reduces the data checking done on the user-supplied column values, and performance is enhanced. This option should only be used when the data being loaded is known to be valid. This may improve the performance of the LOAD process by about 10 or 20 percent.
- Specifying as many devices for the temporary sort directories as you can so that there is opportunity for parallel I/O during sorting. When very large sort areas are required on systems with restricted file sizes then multiple directories are required.
- Specifying as large a value for the sort buffer as possible. The sort buffer memory is allocated from the utility heap. Ensure that the utility heap is defined to be large enough.
- Using `CPU_PARALLELISM` during the LOAD so that you can choose to preserve or not preserve the order of the input data being loaded. The default is to preserve order. Specify `ANYORDER` on the LOAD command to override this default and gain additional performance improvements.

Note: The additional performance improvements would be to the create index time, but may hurt in the query performance time if the original data was in clustered index sequence.

- Using the SORT BUFFER and DATA BUFFER parameters. The SORT BUFFER will improve performance only if indexes are created when using LOAD. The DATA BUFFER parameter specifies the total amount of memory allocated to the LOAD utility as a buffer. The SORT BUFFER parameter specifies the amount of memory allocated for sorting index keys. (This assumes there is real storage to handle the increased buffer size and prevent increased paging.) Both buffer allocations come from the utility heap. You can modify the *util_heap_sz* database configuration parameter accordingly. See "Utility Heap Size (util_heap_sz)" on page 475.
- Specifying multiple directories in different file systems for the USING parameter. Specifying multiple directories improves performance only if indexes are created during the LOAD.
- Controlling the parallelism used during the LOAD in a machine environment where symmetric multi-processor (SMP) exploitation is possible. You can control parallelism in two ways:
  - By using the parameter CPU_PARALLELISM. This parameter controls the degree of paralellism used when parsing, converting, and formatting records.

    Note: When tables include either LOB or LONG VARCHAR data, CPU_PARALLELISM is set to one. Parallelism is not supported in this case.
  - By using the parameter DISK_PARALLELISM. This parameter controls the degree of parallelism used when writing data to the table space containers.

  The maximum value for these parameters will not exceed an internal algorithmic maximum. If the number for these parameters is zero (0), or is not specified, the default is determined at the time the utility is run based on database and system configurations and data characteristics. For more information on the LOAD command, see the *Command Reference* manual.
- Using the BINARY NUMERICS and PACKED DECIMAL parameters when loading binary data in ASC data files, to improve the load time involving numeric data.
- Installing high performance sorting libraries from third party vendors to create indexes during the load. Examples of third party sort products include: SMARTsort** and SyncSort**. This is done through the use of the DB2SORT environment variable. See "Establish Environment Variables and the Profile Registry" on page 60 for more information on environment variables,

The COPY YES/NO option specifies whether to create a copy of the input data during LOAD or not. If "YES" is chosen, performance is reduced because all the data being loaded is copied at the same time. This choice is faster than accepting a **backup pending** state and having to do a backup later before accessing the table. If "NO" is chosen, and forward recovery in enabled, then the table is placed in a **backup pending** state.

LOAD Temporary Space Limitations

When creating indexes during the LOAD, you require at least as much disk space as the sum of the index sizes and possibly twice as much. The space used is temporary; that is, it is located outside the database in the directories specified for the `USING` parameter or in the `tmp` directory defined by the DB2INSTPROF environment variable.

Restarting LOAD and Database Recovery

A LOAD can be restarted following a failure. A copy image of the loaded data can be created for use when recovering a database.

The following discuss these considerations in more detail:

- Restarting the LOAD
- Creating a Copy Image of Loaded Data

Restarting the LOAD

If a failure occurs while loading data, you can restart the load from the last save point or point of consistency; or reload the entire table by using the `REPLACE` option.

The remote file specified in the LOAD restart operation should be the one that was specified in the LOAD command that failed.

There are a number of options available should you decide to restart the load.

If you decide to restart using the `RESTARTCOUNT number` option, then you must use the number of rows at the last successful consistency point. To determine that value, use the LOAD QUERY command with either the name specified with the `REMOTE FILE` option or the default name `db2utmp`. By choosing `RESTARTCOUNT number`, the LOAD restarts from the row following the row identified by the `number` and attempts to finish the load.

Note: The `RESTARTCOUNT number` can only be used with the last successfully completed consistency point. If the last consistency point started but did not complete (that is, SQL3519W is not followed by SQL3520W), then you must carry out the action as described in the help for message SQL3519W.

If you do not want to continue loading rows, or if the failure was during the Build phase, you can use the `RESTARTCOUNT B` option. The LOAD process brings the table to the state of the last save point or point of consistency and then restarts the Build phase. By choosing this option the LOAD restarts, does not attempt to load additional rows, and builds the indexes for the rows already loaded.

If the message file states that the Build phase completed and all temporary files are unmodified as left by the LOAD, you can use the `RESTARTCOUNT D` option. The information on the rows containing duplicate keys stored in the temporary files is used to delete those rows.

The restarted LOAD command should continue until the completion of the LOAD process.

Note: For minor errors such as nonexistent data files or an invalid *dcoldata*, the LOAD will clean up and take the table out of the **load pending** state. You must do the LOAD again in either `REPLACE` or `INSERT` mode with correct parameters.

Creating a Copy Image of Loaded Data

If the table being loaded is part of a recoverable database, then logging is in effect. Since LOAD does not log the changes made to the table, you have the option of specifying `COPY YES` to create a copy of the data being loaded. This copy is used during roll-forward recovery to re-create the changes to the database done by LOAD.

When using this option you should also consider using multiple devices or directories to allow for the best possible I/O exploitation.

For more information on the load copy location file, see “Using the Load Copy Location File” on page 225.

If forward recovery is enabled (`logretain` or `userexit` is “on”) and the `COPY` option was not used, all table spaces in which the loaded table resides are left in a **backup pending** state. A backup of the database or the table space(s) is required to remove this **pending** status. The backup is done before any other units of work against the database or table space can be started.

For more information on how to recover, see Chapter 6, “ Recovering a Database” on page 179.

During forward recovery, if the load copy is not available, then the table spaces (of the loaded table) which are being rolled forward will be set to the **restore pending** state. These table spaces must be restored from a backup image taken after the table load.

An error is returned if you specify `NONRECOVERABLE` and `COPY YES`. There is no need for a copy in such a case since it would not be needed during recovery.

LOAD Exception Table

The exception table is a user-created table which mimics the definition of the table being loaded. It is specified by the `FOR EXCEPTION` option on the LOAD command. The table is used to store copies of rows that violate unique index rules.

Note: Any rows rejected before the building of the index on the loaded table because of invalid data are not inserted into the exception table.

Rows are added to the existing information in the exception table. The existing information may include rows listing check constraint or foreign key violations; or invalid rows from a previous LOAD. If you want only the invalid rows from this LOAD, you will need to remove the existing rows before invoking LOAD.

The exception table used with the LOAD utility is identical to the exception table(s) used by the SET CONSTRAINTS statement. An exception table should be created to perform a load which has a unique index and may have duplicate records. If an exception table is not provided for the LOAD, and duplicate records are found, then the

LOAD will continue. However, only a warning message is issued about the deleted duplicate records and the deleted duplicate records are not placed anywhere.

After completing the load, information in the exception table can be used any way you wish. You may want to use the information to correct any data that was in error and insert the rows into the original table.

The exception table message column has the following structure:

Table 22. Exception Table Message Column Structure for LOAD

Field number	Contents	Size	Comments
1	Number of violations	5 characters	Right justified padded with '0'
2	Type of first violation. Only "I" is used by LOAD	1 character	'I' - Unique Index violation
3	Length of constraint/index token	5 characters	Right justified padded with '0'
4	Constraint/index token	length from the previous field	

Note: Only Unique Index violations will be reported by LOAD. The Check Constraint and Foreign Key violations will be reported by running the `SET CONSTRAINTS` statement with the `IMMEDIATE CHECKED FOR EXCEPTION` options. Only one unique index violation in a row is reported. LONG or Large Object (LOB) data is not inserted into the exception table. Index token is the "IID" value from SYSCAT.INDEXES that identifies the index.

Checking For Constraint Violations

The loaded table may be in the check pending state if it has table check constraints or referential integrity constraints defined on it. The `STATUS` flag of the `SYSCAT.TABLES` in the row corresponding to the loaded table indicates the check pending state of the table. For the loaded table to be usable, the `STATUS` must have a value of "`N`" indicating the normal state of the table.

To remove a table from the check pending state, use the SET CONSTRAINT statement. For more information on the SET CONSTRAINTS statement, see the *SQL Reference* manual. One or more tables may be submitted to be checked in a single invocation. For a dependent table to be checked, the parent table must not be in the check pending state. In the case of a referential integrity cycle, all the tables involved in the cycle must be included in a single invocation of the SET CONSTRAINTS statement.

To manage the loading of several tables, consider the position of each within referential relationships along with the table size and time windows available to carry out the load. It may be convenient, for example, to check the parent table for check constraint violations while the dependent table is loaded. This can only occur if the two tables are not in the same table space.

Exception tables are convenient for a consolidated report of all the rows that have constraints violated. If the exception table option is not used, only the first violation is reported. This may be a cause for frustration when dealing with large tables having more than one constraint violation. The same exception table used for the LOAD utility

may be used for checking constraint violations. As with the LOAD utility, there is no checking done when running the SET CONSTRAINTS statement to ensure that the exception table is empty. The extra timestamp column in the exception table may be used to distinguish newly-inserted rows from the old ones, if necessary.

The SET CONSTRAINTS statement does not activate any `DELETE` triggers as a result of deleting rows that violate constraints. It must be noted, however, that once the table is removed from the check pending state, triggers are active. This implies that, if we correct data and INSERT rows from the exception table into the loaded table, any `INSERT` trigger defined on the table will be activated. The implications of this on the data should be considered and, if necessary, suitable action should be taken. One option is to drop the `INSERT` trigger, `INSERT` rows from the exception table, and then recreate the `INSERT` trigger.

Using the AutoLoader Utility

In a partitioned database, partitioning keys are used to determine the database partition where the data resides. Therefore, data must pass through a *splitting* phase before it can be loaded at the correct database partition.

The entire "split and load" process is accomplished by the AutoLoader utility which uses the hashing algorithm to partition the data into as many output pipes as there are database partitions in the nodegroup in which the table was defined. It then loads these output pipes concurrently across the set of database partitions in the nodegroup. A key feature of the AutoLoader utility, is that it uses pipes for all data transfer required in the split and load process. It also uses multiple database partitions for the splitting phase, thereby improving the performance significantly.

The AutoLoader utility may be run in one of the following modes:

SPLIT_AND_LOAD In this mode, data is split and then loaded on the correct database partitions. Pipes are used for temporary storage and transfer of data.

SPLIT_ONLY With this choice, the data is only split. A set of split data files are generated for the requested output database partitions. You must have sufficient storage for the original input source and for each of the split data files. The output from the split function writes the files in the location pointed to by the parameter `SPLIT_FILE_LOCATION` or in the AutoLoader current working directory. Data is split into separate files that are named using the convention `filename.xxx` where `xxx` is the node number to which the split file belongs and filename is no longer than eight characters.

LOAD_ONLY Data is expected to be already split into separate files that are named using the following convention `filename.xxx` where `xxx` is the node number to which the split file belongs and filename is no longer than eight characters. AutoLoader expects to find these files in the SPLIT_FILE_LOCATION or in the current AutoLoader working directory. These split files are loaded concurrently on their corresponding nodes.

ANALYZE This option generates a customized optimal partitioning map for a nodegroup. It is recommended that a data file with a large number of records be specified as input. The output from the ANALYZE mode can be used with the MAP_FILE_INPUT parameter. The larger the number of records used, the better the representation to the actual data that can be analyzed, and the better the resulting new partitioning map. The map will produce a more even distribution of data across each of the database partitions in the nodegroup.

See Appendix O, “Supplemental AutoLoader Information” on page 851 for more information on database partitions and for platform-specific usage notes. Before using the AutoLoader utility, you should be familiar with the LOAD utility. To learn more about the LOAD process, see “Using the LOAD Utility” on page 141.

Planning to Use the AutoLoader Utility

The AutoLoader utility requires the main autoloader driver *db2autold* and other related executable files each located in the *misc* sub-directory under the *sqllib* sub-directory. A sample configuration file, *autoloader.cfg*, is found under the *autoloader* sub-directory of the *samples* sub-directory of the *sqllib* sub-directory.

Before using the AutoLoader utility, you should:

1. Read Appendix O, “Supplemental AutoLoader Information” on page 851 which shows specific details for using the AutoLoader utility.
2. Create a temporary directory and move the autoloader.cfg file into it. This directory must be accessible from all the participating split and load database partitions.
3. Modify the autoloader.cfg file according to the directions included in the file.
4. You should test the AutoLoader with small amounts of data first to get familiar with the utility.

Running the AutoLoader Utility

The AutoLoader utility is executed by typing the following command:

```
db2autold [options]
```

with one or more of the following options:

-c Uses the config_file specified as the configuration file for the AutoLoader utility. The default is 'autoloader.cfg'.

-d Cleans up the temporary resources allocated by the AutoLoader utility. In case the AutoLoader utility exits abnormally, it is necessary to run this option to clean up all associated temporary directories, files and processes.

-i This makes the cleanup interactive. By default, cleanup is done without a prompt. This must only be used with the -d option.

The AutoLoader utility creates a file called *autoload.log* to keep messages returned from the main autoloader script. You can check the contents of this file to track the progress of the AutoLoader utility. As well, AutoLoader creates files called

load_log.XXX and *splt_log.XXX* which contain messages from the load and split processes respectively.

Some Considerations with AutoLoader

There are some items you should consider before using the AutoLoader utility:

- If you wish to maintain the order of the input data, then only one database partition should be used for splitting. Parallel splitting cannot guarantee that the data is loaded in the order it was received.
- If you are using the LOBSINFILE option of the LOAD utility, then all directories containing the LOB files have to be accessible from all the database partitions being loaded. See Appendix O, “Supplemental AutoLoader Information” on page 851 for information on how this is done.
- For optimal performance, the AutoLoader utility should be invoked on a database partition that is not participating in either the splitting or loading operations. As well, if the splitting nodes are different from the loading database partitions, there is less contention for CPU cycles on any one database partition. On a SMP system, you can improve performance if you ensure at least one CPU for every split to be done.
- If you are using multiple database partitions to split the data and the AutoLoader utility is used with a savecount of greater than zero then an error is returned. This ensures that you can restart the AutoLoader utility since the arrival order of the records cannot be determined when you use multiple splitting database partitions.

Sample AutoLoader Configuration File

```
######################################################################
# Release level of this configuration file.
# Please do not delete or modify this line.
Release=V5.1.0


# LOAD Command
# - Specify a complete LOAD command including the file_name, file_type and
#   the table_name.
# - It may be necessary that the load command is double-quoted if it includes
#   special shell characters, like round brackets ( or ).
#
# - Refer to the Command Reference for the complete syntax.

# DEL data file
# -------------
db2 load from your_data of del replace into your_schema.your_table

# ASC data file
# -------------
# db2 load from your_data of asc modified by reclen=19\
# method L "(1 16, 17 18)" replace into your_schema.your_table
```

```
#####################################################################
#
# Miscellaneous AutoLoader Parameters

# DATABASE ... Name of the database being loaded into.
DATABASE=your_db

# HOSTNAME ... Name of the machine to FTP the data file from. This may be
#              an MVS host or another workstation. Make sure that the .netrc
#              file is set up accordingly. Please comment out with a # or
#              leave blank if file is local.
#HOSTNAME=

# SPLIT_FILE_LOCATION ... The complete path name of the location
#                - to place the split files for SPLIT_ONLY mode
#                - to look for split files if in LOAD_ONLY mode
#                If not specified, and in SPLIT_ONLY mode, the split
#                files are placed in the current working directory.
#                If not specified, and in LOAD_ONLY mode, the
#                AutoLoader utility looks for the split files in the
#                current working directory.
SPLIT_FILE_LOCATION=/u/user/

# OUTPUT_NODES ... Database partitions on which load is to be performed.
#                  The supplied node numbers must be a subset of database
#                  partitions on which the table is defined and must also
#                  exist in the db2nodes.cfg file. If left blank, all
#                  database partitions that the table is defined on will
#                  have data loaded into them.
OUTPUT_NODES=(0,1)

# SPLIT_NODES ... The list of database partitions participating in the
#                 splitting process. Splitting database partitions may be
#                 the same or different from the database partitions being
#                 loaded into. These database partitions must also exist in
#                 the db2nodes.cfg file. If left blank, all database
#                 partitions that the table is defined on will be used for
#                 splitting.
SPLIT_NODES=(0)

# RUN_STAT_NODE ... If "statistics yes" is specified in the LOAD command,
#               then statistics will be collected only on one database
#               partition. This parameter specifies the database partition
#               you wish to collect statistics on. If left blank or -1,
#               the default is the first database partition in output node
#               list.
RUN_STAT_NODE=-1
```

```
#####################################################################
#
# Optional AutoLoader parameters ... These may or may not be specified.

# MODE ... Specify the mode to run AutoLoader in.
#          SPLIT_AND_LOAD is the default.
#
#          Other valid values are:
#          SPLIT_ONLY ... Load process is not performed. Output from the
#                         splitting database partitions is written to files
#                         in the SPLIT_FILE_LOCATION or in the current
#                         AutoLoader working directory.
#
#          LOAD_ONLY  ... Data must be pre-split. The split files are sent to
#                         correct database partition for loading.  The split
#                         filenames must follow the convention filename.xxx
#                         where filename was provided in the LOAD command and
#                         xxx is the nodenumber. Also, it is assumed that
#                         filename.xxx is in the SPLIT_FILE_LOCATION or
#                         in the current AutoLoader working directory.
#
#          ANALYZE    ... This option is used to generate an optimal
#                         partition map for a nodegroup.
MODE=SPLIT_AND_LOAD

# LOGFILE ... This name is used as a base name to create the following files.
#                autoload.LOG  ... used to track progress of AutoLoader.
#                splt_LOG.XXX  ... Holds output from splitting operation on
#                                  database partition xx.
#                load_LOG.XXX  ... Holds output from load operation on
#                                  database partition xx.
LOGFILE=LOG

# NOTNFS_DIR ... The path name to the not-nfs space on each database
#                partition. If not specified, it is "notnfs" with
#                respect to the root directory.
NOTNFS_DIR=/notnfs

#####################################################################
#
# Optional Splitter parameters ... These may or may not be specified.

# CHECK_LEVEL ... Can be either CHECK or NOCHECK
#                 - CHECK: Program will check for truncation of record at
#                   Input/Output (default).
#                 - NOCHECK: Program will not check for truncation of record
#                   at Input/Output.
CHECK_LEVEL=CHECK
```

```
# MAP_FILE_INPUT ... Input filename for the partitioning map. If the
#                    partitioning map is customized rather than a
#                    default one, this parameter must be specified.
#                    It points to the file containing the customized
#                    partitioning map. You can get a customized
#                    partitioning map by either using the db2gpmap program
#                    to extract the map from the database system catalog
#                    table; or you can run the ANALYZE mode db2autold
#                    to generate an optimal map. The map generated by the
#                    ANALYZE mode must be moved to each database partition
#                    in your database before actual loading can proceed.
#
MAP_FILE_INPUT=filename_of_your_customized_partitioning map

# MAP_FILE_OUTPUT ... Output file name for partitioning map. This parameter
#                     should be used with the db2autold program executed
#                     in ANALYZE mode. An optimal partitioning map with even
#                     distribution across all database partitions is
#                     generated. If it is not specified and the running mode
#                     is ANALYZE, a default filename "OutMap" is used.
MAP_FILE_OUTPUT=filename_for_your_optimal_partitioning_map

# TRACE ... Tracing hashing values. Dump of all the data conversion process
#           and output of hashing values. Argument is the number of records
#           to trace.
TRACE=100

# NEWLINE ... Only meaningful if the input data file is an ASC file with each
#             record delimited by a new line character, and the RecLen
#             parameter in the load command is specified.
#             If YES, AutoLoader always checks if the record is terminated
#             by a new line character or not. It also checks if the record
#             length is the same as the expected RecLen or not. The default
#             for this parameter is NO.
NEWLINE=YES
```

Loading into Multiple Database Partitions

If you are loading data into a table in a multiple database partition nodegroup, the LOAD utility requires that the files that are to be loaded were split and contain the correct header information. The LOAD utility verifies the header information that the split operation of AutoLoader writes to each data file to ensure that the data goes to the correct location. (The header information is described in Appendix N, “Splitting Data with db2split” on page 839.)

If you are loading data into a table in a single database partition nodegroup, the files do not have to be split, even if the table is defined to have a partitioning key. In this situation, you would specify the NOHEADER option of the LOAD utility.

The LOAD utility checks that the partitioning map used by the split operation of AutoLoader is the same one specified when the table is being loaded. If not, an error is returned. It also checks that the file partition is loaded at the correct database partition, and that the data types of the partitioning key columns specified during splitting match

the current definition in the catalog. The nodegroup to which the table is loaded cannot be redistributed between the time that the data file is partitioned and the time that the parts are loaded into the corresponding database table. If redistribution has been done, the utility cannot load the partitioned data.

LOAD supports the following flat file formats:

- Non-delimited ASCII (ASC)
- Delimited ASCII (DEL)
- PC/IXF Format (IXF)

However, AutoLoader can only be used to split DEL and ASC files.

Note: IXF files cannot be split, but can be loaded into a single-node nodegroup using the 'NOHEADER' option in the LOAD command.

Using the IMPORT Utility

The IMPORT utility inserts data from an input file into a table or view. If the table or view receiving the imported data already contains data, you can either replace or append the existing data with the data in the file.

Note: If the existing table is a parent table containing a primary key that is referenced by a foreign key in a dependent table, its data cannot be replaced, only added to.

The following is an example of the command line processor syntax for the IMPORT command:

```
db2 import from stafftab.ixf of ixf insert into userid.staff
```

The following information is required when importing data to a table or view:

- The path and the input file name where the data to import is stored.
- The name or alias of the table or view where the data is imported.
- The format of the data in the input file. This format can be IXF, WSF, DEL, or ASC. See “LOAD, IMPORT, and EXPORT File Formats” on page 164 for details.
- Whether the data in the input file is to be inserted, updated, replaced, or appended to the existing data in the table or view.
- A message file name.

When importing into large object (LOB) columns, the data can come either from the same file as the rest of the column data, or from separate files. In the latter case, there is one file for each LOB instance.

The column data in the file contains either the data to load into the column, or a filename where the data to load is stored. The default is the file contains the data to load into the column.

Notes:

1. When LOB data is stored in the file, no more than 32KB of data is allowed. Truncation warnings are ignored.
2. All of the LOB data must be stored in the main file or each LOB stored in separate files. The main file cannot have a mixture of LOB data and file names.

For more information on importing LOBs from files, see the `LOBSINFILE` option in the *Command Reference* manual.

You may also provide the following information:

- The method to use for importing the data: column location, column name, or relative column position.
- The number of rows to INSERT before committing the changes to the table. If you periodically do a COMMIT, this reduces the number of rows that are lost if a failure and a rollback occur during the import.
- The number of records in the file to skip before beginning the import. If an error occurs during an import, you may specify this information to restart the import operation immediately following the last row that was successfully imported and committed.
- The names of the columns within the table or view into which the data is to be inserted.

To import data into a new table, you must have SYSADM authority, DBADM authority, or CREATETAB privilege for the database. To replace data in an existing table or view, you must have SYSADM authority, DBADM authority, or CONTROL privilege for the table or view. To append data to an existing table or view, you must have SELECT and INSERT privileges for the table or view.

With IMPORT, there is a possibility of unequal code page situations involving possible expansion or contraction of the character data. Such situations could occur with Japanese or Traditional-Chinese Extended UNIX Code (EUC) and double-byte character sets (DBCS) which may have different length encodings for the same character. An option, NOCHECKLENGTHS, is used to toggle between two situations:

1. Comparison of input data length to target column length is performed before reading in any data. If the input length is larger than the target, NULLs are inserted for that column if it is nullable. Otherwise, the request is rejected. This is the default.
2. No initial comparison is performed and, on a row-by-row basis, an attempt is made to import the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is imported. Specifying `NOCHECKLENGTHS` will enable this behavior.

The IMPORT utility casts user-defined distinct types (UDTs) to similar base data types automatically. This saves you from having to explicitly cast UDTs to the base data types. Casting allows for comparisons between UDTs and the base data types in SQL.

Use the IMPORT utility to re-create a table that was saved by using the EXPORT utility. The table must have been exported to an IXF file. When creating a table from an IXF

file, not all attributes of the original table are preserved. For example, the referential constraints, foreign key definitions, and user-defined data types are not retained. If the IXF file was created with the LOBSINFILE option, then the length of the original LOB is lost. Attributes of the original table that are preserved or retained are:

- Column information
  - Names
  - Types including user-defined distinct types. (User-defined distinct types are preserved as their base type.)
  - Lengths (except for lob_file types)
  - Codepages (if applicable)
- Index information
  - Name
  - Creator
  - Column names of key parts (with a restriction if + or – are in the names)
  - Ascending or descending
  - Uniqueness

Note: Before running the import utility, you must be connected or connected implicitly to the database into which the data will be imported. Also, the utility issues a COMMIT or ROLLBACK statement; therefore, you should complete all transactions and release all locks by performing either a COMMIT or ROLLBACK before using the utility.

Using IMPORT with Buffered Inserts

In a partitioned database environment, the IMPORT utility can be enabled to use buffered inserts. This reduces the messaging that occurs when data is loaded, resulting in better performance.

To cause the IMPORT utility to use buffered inserts, the BIND command must be used. The import package, db2uimpm.bnd has to be rebound against the database with the INSERT BUF option. This can be achieved using the following commands:

```
db2 connect to your_database

db2 BIND db2uimpm.bnd INSERT BUF
```

However, any one of the individual inserts that are buffered can fail. It is not possible to report the failing row and error back to the user as IMPORT usually does. Therefore, buffered inserts should only be enabled with the IMPORT utility if the user is not concerned about error reporting.

Import in a Client/Server Environment

When you import a file to a remote database, a stored procedure may be called to perform the import on the server. A stored procedure will not be called when:

- The application and database code pages are different
- The file being imported is a multiple-part PC/IXF file
- The method used for importing the data is either column name or relative column position

- The target column list provided is longer than 4K
- An OS/2 or DOS client is importing a file from diskette
- LOBPATHS or LOBSINFILE is specified
- NULL INDICATORS are specified for ASC files.

When importing using a stored procedure, messages are created in the message file using the default language installed on the server. The messages are in the language of the application if the language at the client and the server are the same.

The import utility creates two temporary files in the tmp directory indicated by the DB2INSTPROF environment variable on the database server. One file is for data and the other file is for messages generated by the import utility.

If you receive an error about writing or opening data on the server, make sure that:

- This directory exists
- Sufficient disk space for the files exists
- Write-permission to this directory for the system administrator exists.

Differences Between the IMPORT and LOAD Utilities

This table gives you a quick comparison between the two utilities highlighting the important differences between them.

The IMPORT utility	The LOAD utility
Significantly slower than the LOAD utility on large amounts of data.	Significantly faster than the IMPORT utility on large amounts of data because of LOAD's writing of formatted pages directly into the database.
Limited intra-partition parallelism exploitation.	Exploitation of intra-partition parallelism. Typically, this requires symmetric multiprocessor (SMP) machines.
No FASTPARSE support.	Support for FASTPARSE datatype. Reduced data checking on user-supplied data.
No CODEPAGE support.	Support for CODEPAGE datatype. Converts character data (and numeric data specified in characters) from the code page given with this datatype to the database code page during the load operation.
Creation of table and indexes supported with IXF format.	Table and indexes must exist.
WSF format is supported.	WSF format is not supported.
No BINARYNUMERICS support.	Support for BINARYNUMERICS datatype.
No PACKEDDECIMAL support.	Support for PACKEDDECIMAL datatype.
Can import into views and tables. (Aliases are supported.)	Can load into tables only. (Aliases are supported.)
The table space(s) that the table and its indexes reside in are online for the duration of the import.	The table space(s) that the table and its indexes reside in are offline for the duration of the load.
All rows are logged.	Minimal logging is performed.
Triggers will be fired.	Triggers are not supported.

The IMPORT utility	The LOAD utility
If an import is interrupted and a `commitcount` was specified, the table is usable and will contain the rows that were loaded up to the last commit. The user has the choice to restart the import or use the table as is.	If a load is interrupted and a `savecount` was specified, the table remains in load pending state and cannot be used until the load is restarted to continue the load or the table space is restored from a backup image created some time before the load.
Space required is approximately the size of the largest index plus about 10%. This space requirement is used from the temporary table spaces within the database.	Space required is approximately the sum of the size of all indexes defined on the table and possibly twice this size. The space required is temporary space outside of the database.
All constraints are validated during an import.	Uniqueness is verified during a load but all other constraints must be checked using the SET CONSTRAINTS statement.
The keys of each row are inserted into the index one at a time during import.	During a load, all the keys are sorted and the index is built after the data has been loaded.
If up-to-date statistics are required after an import, RUNSTATS must be run afterwards.	Statistics can be gathered during the load if all the data in the table is being replaced.
You can import into a host database through DB2 Connect.	You cannot load into a host database.
Files that are imported must reside on the node where import is invoked.	Files/pipes that are loaded must reside on the node where the database resides.
No backup image is required.	The backup image can be created during the LOAD procedure.

Using the EXPORT Utility

The EXPORT utility exports data from a database into an operating system file. The output file has the format specified by the data format parameter.

The following is an example of the command line processor syntax for the EXPORT command:

```
db2 export to staff.ixf of ixf  select * from userid.staff
```

The following information is required when exporting data:

- A SELECT statement specifying the data to be exported.
- The path and name of the operating system file that stores the exported data.
- The format of the data in the input file. This format can be IXF, WSF, or DEL. See "LOAD, IMPORT, and EXPORT File Formats" on page 164.
- A message file name.

When exporting from LOB columns, the default action is to select the first 32K bytes of data. The data is placed either in the same file as the rest of the column data, or into separate files. In the latter case, each LOB value is placed in separate files by using the `FILETMOD` option `LOBSINFILE` and the `LOBPATHS/LOBFILE` parameters. For more information, see the *Command Reference*.

Note: Extensions from 000 to 999 are automatically added to the base name given in the `LOBFILE` parameter — one for each LOB file.

You may also provide the following information:

- A method that allows you to specify new column names when exporting to IXF or WSF files. If this method is not specified, the column names from the table or view are used in the exported file.
- A file type modifier to specify additional format information when creating DEL and WSF files.

You must have SYSADM authority, DBADM authority, CONTROL privilege, or SELECT privilege for each table participating in the export.

A table may be saved by using the EXPORT utility and specifying the IXF file format. The saved table may be re-created using the IMPORT utility. The EXPORT utility will fail if the data you want to export exceeds the space available on the file system on which the exported file will be created. In this case, you should limit the amount of data selected by specifying conditions on the `WHERE` clause so that the export file will fit on the target file system. You will have to run the EXPORT utility multiple times to export all the data you desire.

Note: Before running the export utility, you must be connected or connected implicitly to the database from which the data will be exported. Also, the utility will issue a COMMIT statement; therefore, you should complete all transactions and release all locks by performing either a COMMIT or ROLLBACK before calling it.

When you want to use the EXPORT utility in a multiple database partition environment, you can use *db2_all* to have the utility carry out the task at each database partition. The SELECT statement used with EXPORT must be able to only get the data found locally. The selection condition appears as follows:

```
SELECT * FROM tablename WHERE NODENUMBER(column-name) = CURRENT NODE
```

Only the rows from *tablename* found on the local database partition are exported to the *filename* (like *staff.ixf* in the previous example) where there is a file with this name at every database partition. The contents of these files are overwritten by the output from the EXPORT command.

LOAD, IMPORT, and EXPORT File Formats

Four types of files can be imported to a database, and three types can be exported or loaded. The type indicates the format of the data within the operating system file. The supported file formats are:

DEL Delimited ASCII, for exchanging files with a wide variety of industry applications, especially other database products. This is a commonly used way of storing data that separates column values with a special delimiting character.

ASC Non-delimited ASCII for importing or loading data from other applications that create flat text files with aligned column data.

WSF Work-Sheet formats, for exchange with products such as Lotus** 1-2-3** and Symphony**. The LOAD utility does not support this data type. The database manager supports WSF files generated and/or supported by:

- Lotus 1-2-3 Release 1, 1A, 2 and 2J
- Lotus Symphony Release 1.0 and 1.1

IXF PC version of the Integrated Exchange Format, the preferred method for exchange within the database manager. Use PC/IXF to export data from a table so it can be imported later into the same or another table.

For DEL, WSF, and ASC data file formats, define the table, including its column names and data types, before importing the file. The data types in the operating system file fields are converted into the corresponding type of data in the database table. The IMPORT utility accepts data with minor incompatibility problems, including character data imported with possible padding or truncation, and numeric data imported into different types of numeric fields.

For IXF data file formats, the table does not need to exist before beginning the import. It can be automatically created when the data is imported. User-defined distinct types (UDTs) are not made part of the new table column types; instead, the base type is used.

Similarly, when exporting to the IXF data file format, UDTs are stored as base data types in the IXF file.

The following topics describe these file formats:

- Delimited ASCII (DEL) File Format
- Nondelimited ASCII (ASC) File Format
- WSF File Format
- PC/IXF File Format

For more information on using these formats, see the *Command Reference*.

Delimited ASCII (DEL) File Format

A DEL file is a sequential ASCII file with row and column delimiters. It can be used to exchange data with a variety of products using different column delimiters.

Each DEL file is a stream of ASCII characters consisting of cell values ordered by row and then by column. Rows in the data stream are separated by row delimiters. Within a row, the individual cell values are separated by column delimiters. When a file is defined as DEL, spaces that precede the first character or follow the last character of a cell value are discarded.

You can override the default characters for the column delimiter (,), the character string delimiter ("), and the decimal point (.).

The following is an example of a DEL file:

```
"Smith, Bob",4973,15.46
"Jones, Bill",12345,16.34
"Williams, Sam",452,193.78
```

Each line ends with a row delimiter which is the end-of-line indicator used by the operating system. In the case of UNIX-based implementations, the end-of-line indicator is an ASCII line feed (LF) character. In the case of Intel-based implementations, the end-of-line indicator is an ASCII carriage return/line feed (CRLF) sequence. Each line ends with a line feed (LF) character which is the row delimiter. In this example, a row is `"Smith, Bob",4973,15.46`.

Quotes (that is, character string delimiters: ") are required so that the commas in the names are not interpreted as being column delimiters. In the example DEL file above, the first column contains `"Smith, Bob" "Jones, Bill" "Williams, Sam"`.

If you change the column delimiter to a semicolon (;), the character string delimiter to a single quote ('), and the decimal point character to a comma (,), the same file would appear as follows:

```
'Smith, Bob';4973;15,46
'Jones, Bill';12345;16,34
'Williams, Sam';452;193,78
```

When importing or exporting DEL files, keep in mind the following:

- For the character string and column delimiters:
  - A space (X'20') is never a valid delimiter or column delimiter.
  - The period (.) is not a valid character string delimiter, because it conflicts with periods in time and timestamp values.
  - When exporting to a DEL file, for the character delimiter string choose a character that does not occur within the data to be exported. An attempt to export character data containing a character string delimiter will cause a warning message. An attempt to import such a file will produce erroneous results.
- Import of character strings that are not enclosed in character string delimiters is allowed. The end of a *nondelimited* character string is determined by the first occurrence of a space, a character string delimiter, or a row delimiter.
- A null value is indicated by the absence of a cell value where one would normally occur, or by a string of spaces.
- Because some other products restrict the length of character fields, the EXPORT command sends a warning message whenever a character column greater than 254 characters is selected for export. The IMPORT command accommodates fields as long as the longest possible length, which is 32 700 bytes.
- When working with DB2 on Intel-based operating systems, the first occurrence of an end-of-file character (X'1A') that is not within a character string indicates the end of the file. No data following the end-of-file character is imported. If the `NOEOFCHAR` option is specified, this character is ignored.
- Integer, decimal, and scientific notation constants can be imported into numeric database columns that are within the proper range.

- The acceptable forms for importing date and time data are based on the country code of the target database.

When exporting DEL files, all dates by default are in YYYYMMDD format. To get ISO format (YYYY-MM-DD), specify DATEISO in the `FILETMOD` attribute.

Code Page Considerations: When you are importing or exporting a DEL file, the code page for the data is assumed to be the same as that of the application executing the utility. If it is different, unpredictable results may occur. When loading a DEL file, the code page for the data is assumed to be the same as that of the database.

Any graphic data extracted (using EXPORT) by a client running under Japanese or Traditional-Chinese EUC code pages will be encoded using the EUC encoding rather than the UCS-2 internal representation when it is written to the file. Any graphic data imported to (using IMPORT) or loaded by (using LOAD) clients running under these code pages will be converted from the EUC encoding to the UCS-2 internal representation before the data is inserted or loaded, respectively, into the database.

Nondelimited ASCII (ASC) File Format

An ASC file is a sequential ASCII file with row delimiters. It can be used to exchange data with any ASCII product that can create data in a columnar format, including word processors.

Each ASC file is a stream of ASCII characters consisting of data values organized by row and column. Rows in the data stream are separated by a row delimiter, which is the end-of-line indicator used by the operating system. In the case of UNIX-based implementations, the end-of-line indicator is an ASCII line feed (LF) character. In the case of Intel-based implementations, the end-of-line indicator is an ASCII carriage return/line feed (CRLF) sequence. If the `RECLEN=x` option is used, each “x” characters is considered one row.

Each column within a row is defined by a beginning-ending location pair. Each pair represents locations specified as byte positions within a row. (The first position within a row is byte position 1.) The first element of each location pair is the byte within the row where the column begins and the second element is the byte where the column ends. The columns may overlap. Within one ASCII file, every row has the same column definitions.

No special processing is done for column names. Each row is considered to be data, which means that ASC files are assumed to have no row or column names.

See the *API Reference* and the *Command Reference* for more information about ASCII file formats used for import.

Code Page Considerations: When you are importing an ASC file, the code page for the data is assumed to be the same as that of the application executing the utility. If it is different, unpredictable results may occur. When loading an ASC file, the code page for the data is assumed to be the same as that of the database.

Any graphic data extracted (using EXPORT) by a client running under Japanese or Traditional-Chinese EUC code pages will be encoded using the EUC encoding rather than the UCS-2 internal representation when it is written to the file. Any graphic data imported to (using IMPORT) or loaded by (using LOAD) clients running under these code pages will be converted from the EUC encoding to the UCS-2 internal representation before the data is inserted or loaded, respectively, into the database.

WSF File Format

Lotus 1-2-3 and Symphony products use the same basic format, with additional functions added at each new release. The database manager supports the subset of the worksheet records that are the same for all the Lotus products. That is, for the releases of Lotus 1-2-3 and Symphony products supported by the database manager, all file names with any three-character extension are accepted, for example: WKS, WK1, WRK, WR1, WJ2.

Each WSF file represents one worksheet. The database manager uses the following conventions to interpret worksheets and to provide consistency in worksheets generated by its export operations:

- Cells in the first row (ROW value 0) are reserved for descriptive information about the entire worksheet. All data within this row is optional. It is ignored during import.
- Cells in the second row (ROW value 1) are used for column labels.
- The remaining rows are data rows (records, or rows of data from the table).
- Cell values under any column heading are values for that particular column or field.
- A null value is indicated by the absence of a real cell content record (for example, no integer, number, label, or formula record) for a particular column within a row of cell content records.

  Note: A row of all nulls will be neither imported nor exported.

In order to create a file that is compliant with WSF format, some loss of data may occur when exporting from a table into a file with WSF format.

Code Page Considerations: Data in the WSF files use a Lotus code point mapping that is not necessarily the same as existing code pages supported by DB2. As a result, when importing or exporting a WSF file, data is converted from the Lotus code points to/from the code points used by the application code page. DB2 supports conversion between the Lotus code points and code points defined by code pages 437, 819, 850, 860, 863, and 865.

Note: For multi-byte character set users, no conversions are performed.

PC/IXF File Format

The personal computer (PC) version of the IXF format is a specific format used by the database manager. IMPORT and LOAD accept only PC/IXF files, *not* host IXF files. PC/IXF is a structured description of a database table that contains an external representation of the internal table. Data exported in PC/IXF format can be imported into another DB2 for Universal Database product database. The code page value stored in the IXF file must pass code page checks with the application environment and

database. The IMPORT utility can be invoked with the parameter settings indicating that code page mismatches are to be ignored.

Keep the following rules in mind when importing PC/IXF files into tables and views:

- A non-nullable PC/IXF column can be loaded or imported into a nullable column.
- A nullable PC/IXF column can be loaded or imported into a non-nullable column, although some rows may be rejected.
- Numeric columns accept columns of any numeric type, although some data may be rejected because it is out of range.
- Fixed-length string columns in the PC/IXF file that are too long for the target column are not compatible and are not imported or loaded. Variable-length string columns with actual lengths that are not compatible with the target column are processed according to the compatibility rules used when adding data to a table or view. The data is padded on the right with spaces if necessary.
- Date, time, and timestamp columns accept data from PC/IXF columns with matching types and from character PC/IXF columns. Data values from character PC/IXF columns must be valid input values for dates, times, or timestamps for successful insertion into each of the corresponding type columns.
- A file with more than 1024 columns will be rejected.
- Large object (LOB) files can only go into large objects (LOBs).
- Large objects (LOBs) can go into CHAR fields.

Code Page Considerations: A PC/IXF file does not have to be using the same code page as the application running the import or load utility. The code page of the data in the PC/IXF file is stored in the file.

If the PC/IXF file and the application performing the import or load are using the same code page, processing occurs as for a regular application. If they are using different code pages, processing depends on how the import or load utility were invoked:

- If the `FORCEIN` option has been specified, the file code page is ignored and the import or load assume that the data is in the application code page.
- If the `FORCEIN` option is not specified, the results depend on whether a code page conversion table exists for the file code page and the application code page for IMPORT or the database code page for LOAD.
  - If a conversion table exists, the IMPORT utility or LOAD utility converts the data, and the utility continues with a warning that the conversion has occurred.
  - If there is no conversion table, the IMPORT utility or LOAD utility ends with an error.

When exporting a PC/IXF file using the LOBSINFILE option and then importing or loading to a client having a different code page, any CLOBs or DBCLOBs are not converted. The CLOBs and DBCLOBs are kept in separate files when the rest of the data is imported or loaded. To properly import or load CLOB and DBCLOB data, the utility must be used as an application having the same code page as the PC/IXF file.

Any graphic data imported to (using IMPORT) or loaded by (using LOAD) clients running under Japanese or Traditional-Chinese Extended Unix Code (EUC) code pages will be assumed to be encoded using the UCS-2 code set. Mixed character data is

assumed to be encoded using the EUC code set. Similarly, any graphic data extracted (using EXPORT) by clients running under either of the two EUC code pages remains encoded as UCS-2. This is done to improve performance.

Moving Data Between Systems

The IMPORT and EXPORT utilities may be used to transfer data between DB2 databases, and to and from DRDA host databases.

DataPropagator Relational (DPROPR) is another method for moving data between databases in an enterprise.

The following topics provide more information:

- Moving Data Between DB2 Databases
- Moving Data Using the db2move Tool
- Moving Data With DB2 Connect
- Using Replication to Move Your Data

Moving Data Between DB2 Databases

Compatibility considerations are most important when loading or importing/exporting data between Intel-based and UNIX-based platforms.

For more information, see the following topics:

- PC/IXF Format
- Delimited ASCII (DEL) File Formats
- WSF File Format

PC/IXF Format

PC/IXF is the **recommended** format for transferring data between DB2 databases. PC/IXF files allow the Load utility or the Import utility to process numeric data, normally machine dependent, in a machine independent fashion. For example, numeric data is stored and handled differently in Intel** and other hardware architectures.

To provide compatibility of PC/IXF files between all products in the DB2 family the EXPORT utility creates files with numeric data in Intel format, and the IMPORT utility expects it in this format.

Note: Depending on the hardware platform, DB2 products convert numeric values between Intel and non-Intel formats (using byte reversal) during both export and import operations.

Multiple Part Files: UNIX-based implementations of DB2 do not create multiple-part PC/IXF files during export. However, they will allow you to import such a file that was created by DB2 for OS/2. When importing this type of file, all parts should be in the same directory, otherwise an error is returned by the utility.

The single-part PC/IXF files created by the UNIX-based implementations of DB2 export utility can be imported by DB2 for OS/2.

Delimited ASCII (DEL) File Formats

DEL files have differences based on the operating system on which they were created. The differences are:

- Row separator characters
  - Intel-based text files use a carriage return/line feed (CRLF) sequence
  - UNIX-based text files use a line feed (LF) character
- End-of-file character
  - Intel-based text files have an end-of-file character (X'1A')
  - UNIX-based text files do **not** have an end-of-file character

Since DEL export files are text files, they may be transferred from one operating system to another. File transfer programs handle the above differences if you transfer the file using the text mode. Using the binary mode to transfer the file does **not** convert row separator and end-of-file characters as required.

If character data fields contain row separator characters, these will also be converted during the file transfer. This conversion will cause an inappropriate change to the data and as a result, when the file is imported into a database on the different platform, data shrinkage or expansion may occur. For this reason, we recommend that you **do not** use DEL export files to move data between DB2 databases.

WSF File Format

Numeric data in WSF format files is stored using Intel machine format. This format allows Lotus WSF files to be transferred and used in different Lotus operating environments (for example, Intel-based and UNIX-based systems).

As a result of this consistency in internal formats, exported WSF files from DB2 products can be used by Lotus 1-2-3 and Symphony running on a different platform. DB2 products can also import WSF files that were created on different platforms.

Transfer WSF files between operating systems platforms in binary, not text mode.

Do not use the WSF file format to transfer data between DB2 databases, since a loss of data may occur. Use the PC/IXF file format instead.

Moving Data Using the db2move Tool

`db2move` is a tool that can help move large numbers of tables between DB2 databases located on workstations. `db2move` queries the system catalog tables for a particular database and compiles a list of all user tables. The tool then exports these tables in PC/IXF format. The PC/IXF files can be imported or loaded to another local DB2 database on the same system, or can be transferred to another workstation platform and imported or loaded to a DB2 database on that platform.

`db2move` calls the DB2 export, import, and load APIs depending on the action requested by the user. Therefore, the requesting user ID must have the correct authorization required by the APIs or the request will fail. Also, `db2move` inherits the limitations and restrictions of the APIs. `db2move` is found in the `misc` subdirectory of the `sqllib` directory.

The syntax of the tool is:

```
db2move dbname action [options...]
```

The `dbname` is the name of the database. The `action` must be one of: EXPORT, IMPORT or LOAD. The `options` are:

-tc table-creators. The default is all creators.

This is an EXPORT action only. If specified, only those tables created by the creators listed with this option are exported. If not specified, the default is to use all creators. When specifying multiple creators, each must be separated by commas; no blanks are allowed between creator IDs. The maximum number of creators that can be specified is 10. This option can be used with the "-tn" table-names option to select the tables for export.

The wildcard character, asterisk (*), can be used in table-creators and can be placed anywhere in the string.

-tn table-names. The default is all user tables.

This is an EXPORT action only. If specified, only those tables whose names match exactly to those in the specified string are exported. If not specified, the default is to use all user tables. When specifying multiple table-names, each must be separated by commas; no blanks are allowed between table-names. The maximum number of table-names that can be specified is 10. This option can be used with the "-tc" table-creators option to select the tables for export. `db2move` will only export those tables whose names are matched with specified table-names and whose creators are matched with specified table-creators.

The wildcard character, asterisk (*), can be used in table-names and can be placed anywhere in the string.

-io import-option. The default is REPLACE_CREATE.

Valid options include INSERT, INSERT_UPDATE, REPLACE, CREATE, and REPLACE_CREATE.

-lo load-option. The default is INSERT.

Valid options include INSERT and REPLACE.

-l lobpaths. The default is the current directory.

This option shows the absolute path names where LOB files are created (as part of EXPORT) or searched for (as part of IMPORT or LOAD). When specifying multiple lobpaths, each must be separated by commas; no blanks are allowed between lobpaths. If the first path runs out of space (during EXPORT) or the files are not found in the path (during IMPORT or LOAD), the second path will be used. Each subsequent path will be used for the same reasons should the same conditions exist.

If the action is an EXPORT and lobpaths are specified, **all files in the lobpath directories are deleted**, the directories are removed, and new directories are created. If not specified, the current directory is used for the lobpath.

-u userid. The default is the logged on user ID.

Both user ID and password are optional. However, if one is specified, **both** must be specified. If db2move is run on a client connecting to a remote server, user ID and password should be specified.

-p password. The default is the logged on password.

Both user ID and password are optional. However, if one is specified, **both** must be specified. If db2move is run on a client connecting to a remote server, user ID and password should be specified.

The following are several examples showing the db2move:

- db2move sample export

  This will export all tables in sample; the defaults are used for all options.
- db2move sample export -tc userid1,us*rid2 -tn tbname1,*tbname2

  This will export all tables created by "userid1" or user IDs LIKE "us%rid2"; and, table-name is "tbname1" or table-names LIKE "%tbname2."
- db2move sample import -l D:\LOBPATH1,C:\LOBPATH2

  This example is applicable for Intel-based platforms only. This will import all tables in sample; any LOB files are to be searched for using lobpaths "D:\LOBPATH1" and "C:\LOBPATH2."
- db2move sample load -l /home/userid/lobpath,/tmp

  This example is applicable for UNIX-based platforms only. This will load all tables in sample; any LOB files are to be searched for using the lobpath subdirectory in the userid subdirectory of the the home directory or in the tmp subdirectory.
- db2move sample import -io replace -u userid -p password

  This will import all tables in sample in REPLACE mode; the user ID and password are used.

Usage notes:

1. This tool exports, imports, or loads user-created tables. If you want to duplicate a database from one platform to another platform db2move only helps you to move the tables. You need to consider moving all other objects associated with the tables such as: aliases, views, triggers, user-defined functions, and so on. db2look can help you move some of these objects by extracting the data definition language (DDL) statements from the database. db2look is another tool that is found under the misc subdirectory in the sqllib subdirectory.
2. When EXPORT, IMPORT, or LOAD APIs are called by db2move, the FileTypeMod parameter is set to "lobsinfile." That is, LOB data is kept in separate files from PC/IXF files. There are 26 000 file names available for LOB files.
3. LOAD action must be run locally on the machine where the database and data file reside. When the LOAD API is called by db2move, the CopyTargetList parameter is set to NULL. That is, no copying is done. If logretain is on, the LOAD cannot be rolled forward later on. The table space where the loaded tables reside is placed in "backup pending" state and is not accessible. A full database backup or a table space backup is required to take the table space out of the "backup pending" state.

The db2move LOAD action is not supported in DB2 Universal Database where partitioned databases may be used.

Notes when using EXPORT:

- Input: None.
- Output:

  EXPORT.out The summarized result of the EXPORT action.

  db2move.lst The list of original table names, their corresponding PC/IXF file names (tabnnn.ixf), and message file names (tabnnn.msg). This list, the exported PC/IXF files, and LOB files (tabnnnc.yyy) are used as input to the db2move IMPORT or LOAD action.

  tabnnn.ixf The exported PC/IXF file of a table.

  tabnnn.msg The export message file of the corresponding table.

  tabnnnc.yyy The exported LOB files of a table.

  "nnn" is the table number. "c" is a letter of the alphabet. "yyy" is a number ranging from 001 to 999.

  These files are created only if the table being exported contains LOB data. If created, these LOB files are placed in the "lobpath" directories. There are a total of 26 000 possible names for the LOB files.

  system.msg The message file containing system messages for creating or deleting file or directory commands. This is only used if the action is EXPORT and a lobpath is specified.

Notes when using IMPORT:

- Input:

  db2move.lst An output file from the EXPORT action.

  tabnnn.ixf An output file from the EXPORT action.

  tabnnnc.yyy An output file from the EXPORT action.

- Output:

  IMPORT.out The summarized result of the IMPORT action.

  tabnnn.msg The import message file of the corresponding table.

Notes when using LOAD:

- Input:

  db2move.lst An output file from the EXPORT action.

  tabnnn.ixf An output file from the EXPORT action.

  tabnnnc.yyy An output file from the EXPORT action.

- Output:

  LOAD.out The summarized result of the LOAD action.

  tabnnn.msg The LOAD message file of the corresponding table.

Moving Data With DB2 Connect

You may be working in a more complex environment where you need to move data between a host database system and the workstation environment. In such an environment, you may work with DB2 Connect; as the gateway for the data from the host to the workstation as well as the reverse.

The following section discusses the considerations when importing and exporting data using DB2 Connect.

Using Import and Export Utilities

The import and export utilities let you move data from a DRDA server database to a file on the DB2 Connect workstation or vice versa. You can then use this data with any other application or RDBMS that supports this import/export format. For example, you can export data from DB2 for MVS/ESA into a delimited ASCII file and later import it into a DB2 for OS/2 database.

You can perform export and import functions from a database client or from the DB2 Connect workstation.

Notes:

1. The data to be imported or exported must comply with the size and data type restrictions of both databases.
2. To improve import performance, you can use compound SQL. Specify `COMPOUND=`*`number`* in the import API or the CLP *filetype-mod* string parameter to group the specified number of SQL statements into a block. This may reduce network overhead and improve response time.
3. For information on the syntax of the import and export utilities from the command line processor, see the *Command Reference* manual.

Moving Data from a Workstation to a DRDA Server: To export to a DRDA server database:

1. Export the rows of information from the DB2 table into a PC/IXF file.
2. If the DRDA server database does not contain a table having attributes compatible with the information to be imported into it, create a compatible table.
3. Using the INSERT option, import the PC/IXF file to a table in the DRDA server database.

Moving Data from a DRDA Server to a Workstation: To import data from a DRDA server database:

1. Export the rows of information from the DRDA server database table to a PC/IXF file.
2. Use the PC/IXF file for importing to a DB2 table.

Restrictions: With the DB2 Connect program, import or export operations must meet the following conditions:

- The file type must be PC/IXF.
- Index definitions are not stored on export or used on import.
- A table with attributes that are compatible with those of the data must exist before you can import to it. Importing through the DB2 Connect program cannot create a table because INSERT is the only supported option.
- A commit count interval must not be specified with import.

If these conditions are violated, the operation will fail and an error message will be generated.

Mixed Single-Byte and Double-Byte Data: If you import and export mixed data (columns containing both single-byte and double-byte data), consider the following:

- On systems that store data in EBCDIC (MVS, OS/390, OS/400, VM, and VSE), shift-out and shift-in characters mark the start and end of double-byte data. When you define column lengths for your database tables, be sure to allow enough room for these characters.
- Variable-length character columns are recommended unless the data in a column has a consistent pattern. If it does, fixed length is acceptable.

Replacement for SQLQMF Utility: The function of the SQLQMF utility with DDCS for OS/2 has been replaced by the DB2 Connect Import/Export functions. The advantages are:

- No need for QMF on the host
- No need to logon to the host (a TSO id is still required on DB2 for MVS/ESA or DB2 for OS/390)
- Supports DB2 for MVS, DB2 for OS/390, DB2 for OS/400, and DB2 for VM and VSE
- Good performance achieved by using compound SQL
- Supports several file formats, in addition to ASCII
- Can be run from a client machine with no SNA connectivity.

Refer to the *Command Reference* for further information on using these commands.

Using Replication to Move Your Data

Replication allows you to copy data on a regular basis to multiple remote databases. If you need to have updates to a master database automatically copied to other databases, you can use the replication features of DB2 to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied. The replication features in DB2 are a part of a larger IBM solution for replicating data in small and large enterprises—IBM Relational Data Replication (IBM Replication).

The IBM Replication tools are a set of IBM DataPropagator Relational (DPROPR) programs and DB2 Universal Database tools that copy data between distributed relational database management systems:

- Between DB2 Universal Database platforms.
- Between DB2 Universal Database platforms and host databases supporting Distributed Relational Database Architecture (DRDA) connectivity.
- Between host databases that support Distributed Relational Database Architecture (DRDA) connectivity.

Based on the DPROPR V1 offering, IBM Replication tools allow you to copy data automatically between DB2 relational databases, as well as nonrelational and non-IBM databases.

You can use the IBM Replication tools to define, synchronize, automate, and manage copy operations from a single control point for data across your enterprise. The replication tools in DB2 Universal Database offer replication between relational databases only. The tool set manages the copying (replication) of data in a store-and-forward manner.

Why use Replication?

Replication allows you to give end-users and applications access to production data without putting extra load on the production database. You can copy the data to a database local to an end-user or application, rather than have them access the data remotely. A typical replication scenario involves a source table with copies in one or more remote databases, for example, a central bank and its local branches. A change occurs in the “master” or source database. At a predetermined time, an automatic update of all of the other DB2 relational databases takes place and all the changes are copied to the target database tables.

The replication tools allow you to customize the copy table structure. You can use SQL when copying to the target database to subset, aggregate, or otherwise enhance the data being copied. You can also create the copy tables structure to fit your needs: read-only copies that duplicate the source table, show data at a certain point in time, provide a history of changes, or stage data to be copied to additional target tables. Additionally, you can create read-write copies that can be updated by end-users or applications and have the changes replicated back to the master table. You can replicate views of source tables and views of copies. Event-driven replication is also possible.

The replication tools currently support DB2 on MVS/ESA, AS/400, AIX, OS/2, VM and VSE, Windows NT, HP, and the Solaris Operating environment. You can also replicate to non-IBM databases, such as Oracle, Microsoft SQL Server, and Lotus Notes.

The IBM Replication Tools in Detail

There are two components of the IBM Replication tools solution: IBM DPROPR Capture and IBM DPROPR Apply. You can setup these two components with the DB2 Control Center. The operation of these two components, and the monitoring of them, happen outside of the Control Center.

The IBM DPROPR Capture program captures the changes from the source tables. A source table can be an external table containing SQL data from a file system or nonrelational database manager loaded outside DPROPR; an existing table in the database; or, a table that has previously been updated by the IBM DPROPR Apply program, which allows changes to be copied back to the source or to other target tables.

The changes are copied into a change data table, where they are stored until the target system is ready to copy them. The Apply program then takes the changes from the change data table and copies them to the target tables.

You use the Control Center to set up the replication environment, define source and target tables, specify the timing of the automated copying, specify SQL enhancements to the data, and define relationships between the source and the target tables.

For more information, see the *Replication Guide and Reference*, S95H-0999.

Chapter 6. Recovering a Database

A database can become unusable because of hardware or software failure (or both), and the different failure situations may require different recovery actions. You should have a strategy in place to protect your database against the possibility of these failure situations. When designing a strategy, you should also rehearse it. This will allow you to detect any shortcomings in the plan, and to avoid problems when you have to recover the database.

This chapter discusses the different recovery methods that can be used in the event there is a problem involving the database. Also discussed are considerations and decisions that will assist in determining the recovery method best suited to your business environment. Each recovery method is described along with the associated concepts, and the commands provided with the product to support these methods.

The following are major topics within this chapter:

- Overview of Recovery
- Factors Affecting Recovery
- Recovery Method: Crash Recovery
- Recovery Method: Restore Recovery
- Recovery Method: Roll-Forward Recovery
- ADSTAR Distributed Storage Manager

One type of problem that requires database recovery is not handled by DB2: the corruption of data that is caused by errant logic or incorrect input in an application. You can use restore and roll-forward recovery to recover the database to a point in time that is close to when the application began working with the database. Or, you can attempt to back out the effects of the application on the database by executing the transactions in reverse. You must exercise caution if you decide to follow the second approach. This chapter does not provide further description about application errors.

Overview of Recovery

You need to know the strategies available to you to help when there are problems with the database. Typically you will deal with media and storage problems, power interruptions, and application failures. You need to know that you can back up your database, or individual table spaces, and then rebuild them should they be damaged or corrupted in some way. The rebuilding of the database is called *recovery*. There are three ways recovery of a damaged database can take place: *crash recovery*, *restore*, and *roll-forward*.

1. Crash recovery is the method that protects a database from being left in an inconsistent, or unusable, state. Transactions, or units of work, against the database can be interrupted unexpectedly. For example, should a failure (power interruption, application failure) occur before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state.

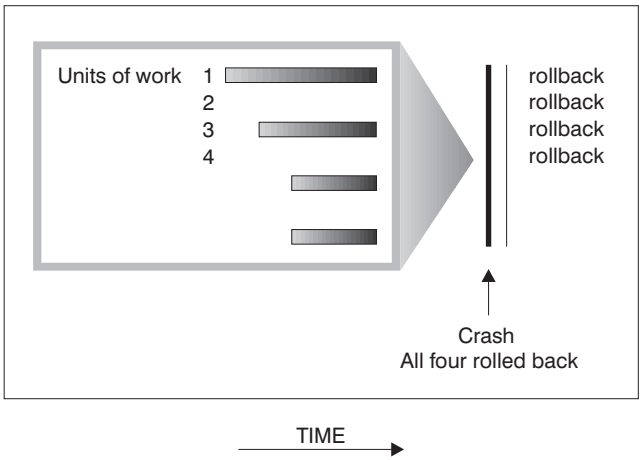


Figure 20. Rolling Back Units of Work

The database then needs to be moved to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

You can do this by entering a RESTART DATABASE command. If you want this done in every case of a failure, then you should consider the use of the **automatic restart enable** *(autorestart)* configuration parameter. The default for this configuration parameter is that the RESTART DATABASE routine will be started every time it is needed. When *(autorestart)* is enabled, the next connect request to the database after the failure causes RESTART DATABASE to be executed.

Crash recovery always moves the database to a consistent and usable state. If crash recovery occurs for a database that is enabled for forward recovery (that is, the *logretain* or *userexit* configuration parameter is on for the database), and an error occurs during crash recovery that is attributable to an individual table space, that table space is taken off-line. Crash recovery continues. The table space taken off-line is placed in a **roll-forward pending** state.

At the completion of crash recovery, the other table spaces in the database are still usable and connections to the database can be established. (There are exceptions involving the table spaces that have the temporary tables or the system catalog tables. These will be discussed under roll-forward recovery.)

Following crash recovery, you may need to take additional action. You may need to work with the table spaces taken off-line as mentioned above. You may need to conduct a restore recovery and/or a roll-forward recovery depending on the error.

2. Restore recovery (also known as **version control**) allows for the restoration of previous version or image of the database that was made using the BACKUP command.

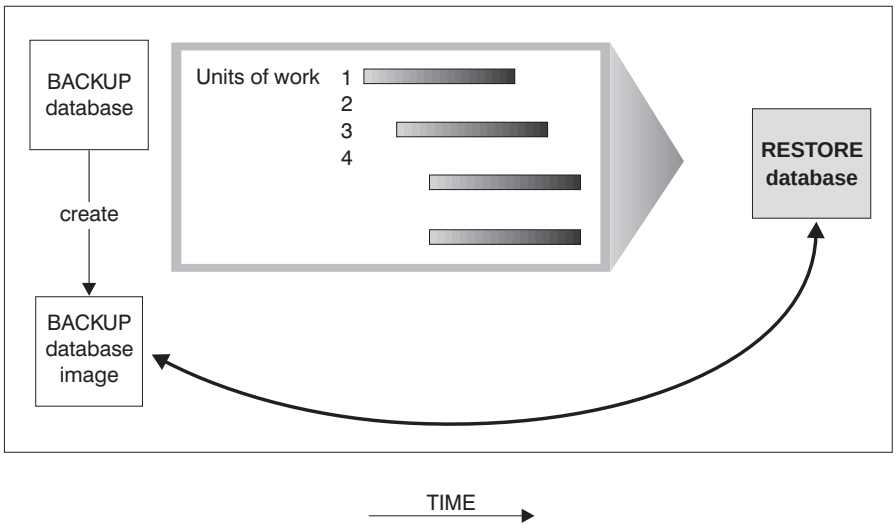


Figure 21. Restoring a Database

A *database restore* will rebuild the entire database using a backup of the database made at some point earlier. A backup of the database allows you to restore a database to a state identical to the time when the backup was made. Every unit of work from the time of the backup to the time of the failure is lost. (The need to re-create these units of work introduces the possibility of the next recovery method, roll-forward recovery, which is discussed later.)

Using the database restore recovery method, you must schedule and perform a full backup of the database on a regular basis.

In a partitioned database environment, the database is located across many database partition servers (or nodes). You must restore all database partitions, and the backups that you use for the RESTORE must all have been taken at the same time. (Each database partition is backed up and restored separately.) A backup of each database partition taken at the same time is known as a *version backup*.

3. Roll-forward recovery may be the next task to be done following a restore depending on the state of the database. For roll-forward recovery to be possible on a database, the database must be recoverable, and must be in the roll-forward pending state at the end of the restore.

   Recoverable databases have either the *logretain* or *userexit* (or both) database configuration parameters turned “on.” This allows for active and archived logs to be kept and results in the ability for the database to have roll-forward recovery. Table space BACKUP and RESTORE, and online BACKUP and RESTORE, are applicable to recoverable databases only.

   Non-recoverable databases have both *logretain* and *userexit* turned “off.” Only active logs are kept for crash recovery; no roll-forward recovery is allowed. Restore

recovery using offline backups is the primary means of recovery for problems with this mode of database.

The scenarios that you need consider at this point are:

a. *Database roll-forward recovery*, which follows the restore of the database with the application of database logs. The database logs record all changes made to the database. This method completes the recovery of the database to a state identical to the time just before the failure.

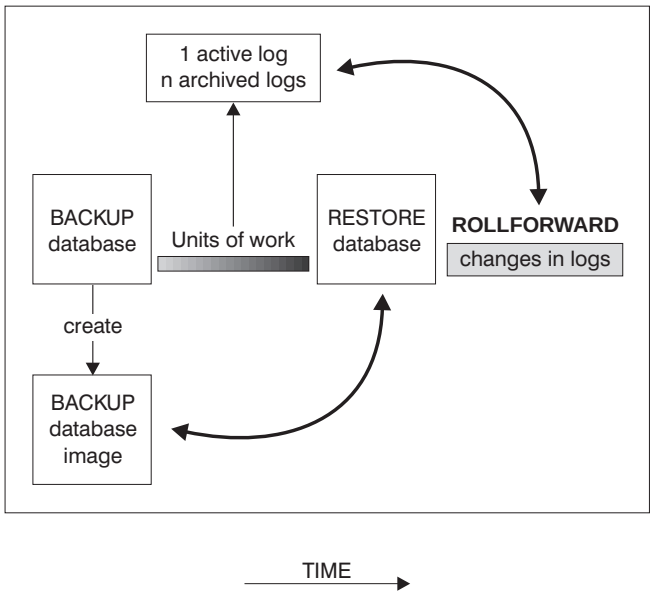


Figure 22. Database Roll-Forward Recovery

To use the database roll-forward recovery method, you must have created a backup of the database as well as archiving the logs by enabling either *logretain* or *userexit*. There are decisions that you must make regarding the logging procedure that you use. (Logging is discussed in more detail later. See the section on “Database Logs” on page 185.)

In a partitioned database system, the database is located across many database server partitions. In this environment, if you are performing point-in-time roll-forward recovery, all database partitions must be rolled forward to ensure that all partitions are at the same level. If you need to restore a single database partition, you can perform roll-forward recovery to the end of the logs to bring it up to the same level as the other database partitions in the database.

b. When the database is enabled for forward recovery, it is also possible to back up and restore table spaces. *Table space restore* requires a backup made using BACKUP. This backup can be of the entire database (all of the table

spaces) or of one or more individual table spaces. This method restores the selected table spaces to a state identical to the time the backup was made.

Notes:

1) Those table spaces not selected at the time of the BACKUP will not be in the same state as those that were restored.

2) Using the table space restore recovery method, you must identify “key” table spaces in the database to be recovered as well as schedule and perform a backup of the database or the “key” table spaces on a regular basis.

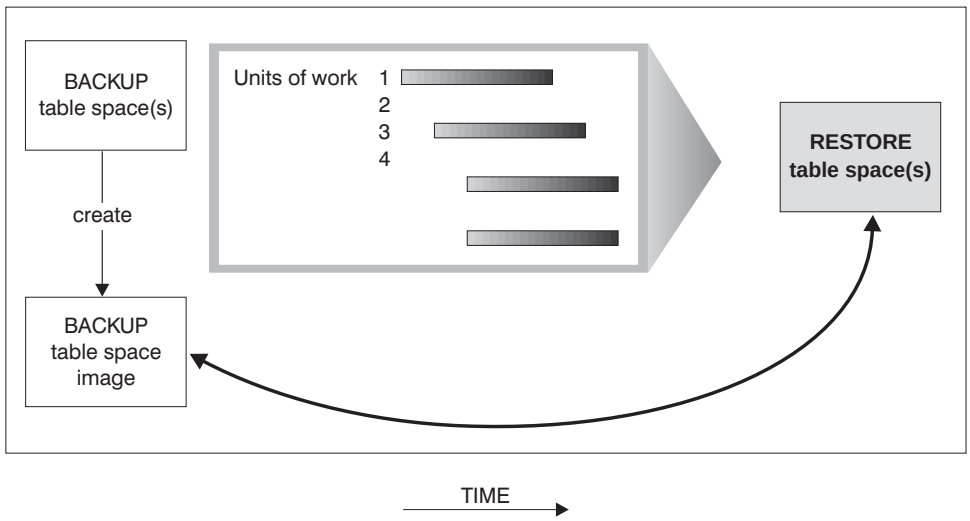


Figure 23. Restoring One or More Table Spaces

Table space roll-forward recovery can be required in the following two situations:

1. If one or more table spaces are in a **roll-forward pending** state because of crash recovery, you can use the ROLLFORWARD command to apply the logs against the table spaces.

   Note: If the table space in error contains the system catalog tables, you will not be able to start the database. You must restore the SYSCATSPACE table space, then perform roll-forward recovery on it to the end of the logs.

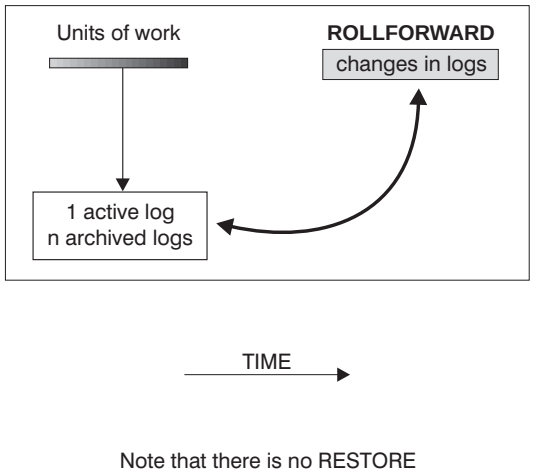


Figure 24. Table Space Roll-Forward Recovery

2. The second scenario occurs after a table space restore. A table space is always in the roll-forward pending state after it is restored, and it must be rolled forward. Again, use the ROLLFORWARD command to apply the logs against the table spaces.

In a partitioned database system, if you are rolling forward a table space to a point in time, you do not have to supply the list of nodes (database partitions) on which the table space resides. The database manager submits the ROLLFORWARD request to all database partitions. If you are rolling forward a subset of the table spaces to the end of the logs, you must supply the list of nodes. If you want to roll forward all table spaces to the end of the logs, you do not have to supply the list of nodes. By default, the ROLLFORWARD request is sent to all database partitions.

Factors Affecting Recovery

To decide which database recovery method to use, you must consider the following key factors:

- Will the database be recoverable or non-recoverable?
- How near to the time of failure will you need to recover the database (the point of recovery)?
- How much time can be spent recovering the database? This would include:
  - Time between backups (will affect roll-forward recovery)?
  - Time the database is usable or accessible (backing up online or offline based on data availability needs)?
- How much storage space can be allocated for backup copies and archived logs?
- Will you be using table space level or full database level backup?

In general, a database maintenance and recovery strategy should ensure that all information is available when it is required for database recovery. The strategy should include a regular schedule for taking database backups, as well as scheduled backups when a database is created, or in the case of a partitioned database system, when the system is scaled by adding or dropping database partition servers (nodes). In addition to these basic requirements, a good strategy will include elements that reduce the likelihood and impact of database failure.

The following topics provide additional information:

- Recoverable and Non-Recoverable Databases
- Database Logs
- Reducing Logging on Work Tables
- Point of Recovery
- Frequency of Backups and Time Required
- Recovery Time Required
- Storage Considerations
- Keeping Related Data Together
- Recovery Performance Considerations
- Disaster Recovery Considerations

Recoverable and Non-Recoverable Databases

If you can re-create data easily, the database holding that data is a candidate to be a non-recoverable database. For example:

- Tables that hold data from an outside source that is used for read-only applications (and the data is not mixed with existing data) should be considered for placement within a non-recoverable database.
- Tables with small amounts of data. Here recovery is not a problem. Rather, there is just not enough logging done for the data to justify the added complexity of managing log files and rolling forward after a restore.
- Large tables where small numbers of rows are periodically added. Again, there is not enough volatility to justify managing log files and rolling forward after a restore.

If you cannot re-create data easily, then the database holding that data is a candidate to be a recoverable database. The following are examples of data that should be part of a recoverable database:

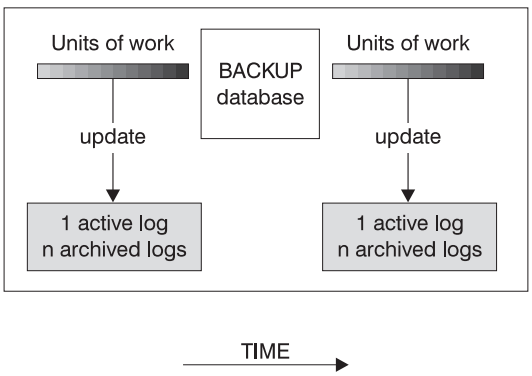
- Data that you cannot re-create. This includes data whose source is destroyed after the data is loaded, and data that is manually entered into tables.
- Data that is modified by application programs or workstation users after it is loaded into the database.

Database Logs

All databases have logs associated with them. These logs keep records of database changes. Some logs, called *active* logs, are used by crash recovery to prevent a failure (system power, application error) from leaving a database in an inconsistent state. Changes already made but not committed because of the failure are rolled back. All committed units of work, which may not have been physically written to disk because of the failure, are redone. These actions ensure the integrity of the database.

Roll-forward recovery can use both the active logs and logs that have been *archived* to rebuild a database either to the end of the logs, or to a specific point in time. The roll-forward function achieves this by reapplying changes that are found in the archived and active logs to the restored database.

Roll-forward recovery can also use logs to rebuild a table space. You can recover a table space to the end of the logs, or to a specific point in time.



Logs are used between backups to track the changes to the databases.

Figure 25. Active and Archived Database Logs in Roll-forward Recovery

The types of database logs that pertain to recovery follow:

Active logs

Active logs contain transactions that have been committed, but may not have been physically written from memory (buffer pool) to disk (database containers). These logs contain information necessary to roll-back any active transaction not committed during normal processing. The RESTART DATABASE command uses the active logs, if needed, to move the database to a consistent and usable state by means of crash recovery. The ROLLFORWARD command may also use the active logs, if needed, during a point-in-time recovery or a recovery to the end of the logs. Active logs are located in the database log path directory.

Online archived logs

When all changes in the active log are no longer needed for normal processing, the log is closed, and becomes an archived log. An archived log is said to be *online* when it is stored in the database log path directory.

Offline archived logs

You also have the ability to store archived logs in a location other than the database log path directory, by using a user exit program. (See Appendix J, “User Exit for Database Recovery” on page 753 for information.) An archived log is said to be *offline* when it is not stored in the database log path directory.

Two parameters in the database configuration file allow you to change where archived logs are stored: the *newlogpath* parameter and the *userexit* parameter. Changing the *newlogpath* parameter also affects where active logs are stored.

To determine which log extents in the database log path directory are archived logs, check the value of the database configuration file parameter *loghead*. This parameter indicates the lowest numbered log that is active. Those logs with sequence numbers less than that of this log are archived logs and can be moved.

Notes:

1. If you erase an active log, the database becomes unusable and must be restored before it can be used again. Also, you will be able to roll forward the changes from the logs only up to the first log that was erased.
2. If you are concerned that your active logs may be damaged (due to a disk crash), you should consider mirroring the volumes on which the logs are stored. By having multiple copies of the logs, you will not lose any transactions, which may happen when active logs are damaged.

Reducing Logging on Work Tables

If your application creates and populates work tables from master tables, and you are not concerned about the recoverability of these work tables because they can be easily re-created from the master tables, you may want to create the work tables with the NOT LOGGED INITIALLY parameter of the CREATE TABLE statement. The advantage of using the NOT LOGGED INITIALLY parameter is that any changes made on the table (including Insert, Delete, Update, or Create Index operations) in the same unit of work that the table was created in will not be logged. This not only reduces the logging that is done, but also obtains better performance for your application.

Notes:

1. You can create more than one table with the NOT LOGGED INITIALLY parameter in the same unit of work.
2. Changes to the catalog tables and other user tables are still logged.

Because changes to the table are not logged, you should consider the following when deciding to use the NOT LOGGED INITIALLY parameter:

- All changes to the table are written out at commit time. This means that the commit may take longer.
- An error received for any operation in a unit of work in which the table is created will result in the rollback of the entire unit of work. In this case, the application receives the SQLCODE -1476 (SQLSTATE 40506).
- When rolling forward, you cannot recover these tables. If the roll-forward operation encounters a table that was created with the NOT LOGGED INITIALLY parameter, this table will be marked as unavailable. After the database is recovered, any attempt to access the table will result in SQL1477N being returned.

  Note: When a table is created, row locks are held on the catalog tables until a COMMIT is done. To take advantage of the no logging behavior, you must

populate the table in the same unit of work in which it is created. This has implications for concurrency. For more information, see “Concurrency” on page 265.

For more information about creating tables, see the *SQL Reference* manual.

Point of Recovery

The restore and roll-forward methods provide different points of recovery. The restore-only method involves making an offline, full database backup copy of the database at scheduled times. With this method, the backup copy of the database is only as current as the time that the last backup was made. For instance, if you make a backup copy at the end of each day and you lose the database midway through the next day, you will lose a half-day's worth of changes.

In the roll-forward recovery method, changes made to the database are retained in logs. With this method, you first restore the database or table space(s) using a backup copy; then you use the logs to reapply changes that were made to the database since the backup copy was created.

With roll-forward recovery enabled, you can take advantage of online backup and table space level backup. For full database and table space roll-forward recovery, you can choose to recover to the end of the logs or to a specified point-in-time. For instance, if an application corrupted the database, you could start with a restored copy of the database, and roll-forward changes up until just before that application started. All units of work in the logs after the time specified will not be reapplied.

You can also roll forward table spaces to the end of the logs, or to a specific point in time. For more information about rolling forward table spaces, see “Rolling Forward Changes in a Table Space” on page 220.

Frequency of Backups and Time Required

Your recovery plan should allow for regularly scheduled backups, since backing up a database requires time and system resource.

You should take full database backups regularly, especially if you archive the logs (which allows for roll-forward recovery). If your recovery strategy includes roll-forward recovery, a recent full database backup will mean that there are fewer archived logs to apply to the database, which reduces the amount of time required by the ROLLFORWARD utility to recover the database.

You should also consider that instead of overwriting backups and logs, that you save more than one full database backup and its associated logs as an extra precaution.

You can do a backup while the database is either *online* or *offline*. If it is *online*, other applications or processes can continue to connect to the database as well as read and modify data while the backup task is running. If the backup is performed *offline*, only the backup task can be connected to the database. The implication of offline backup is

that the rest of your organization cannot connect to the database while the backup task is running.

To reduce the time when the database is not available, consider using online backups. Online backups are supported only if roll-forward recovery is enabled. If roll-forward recovery is enabled and you have a complete set of logs, you can rebuild the database should the need arise.

If you use DMS table spaces, you can store different types of data in their own table spaces to reduce the time required for backups. You can keep table data in one table space, the LONG and LOB data in another table space, and the INDEX data in another table space.

If you reorganize a table, you should back up the affected table spaces after the operation completes. If you have to restore the table spaces, you will not have to roll forward through the data reorganization.

If a database contains large amounts of long field and LOB data, backing up the database could be very time-consuming. The BACKUP command provides the capability to back up selected table spaces. By storing long field and LOB data in separate table spaces, the time required to complete the back up of the data can be reduced by choosing not to back up the table spaces containing the long field and LOB data. If the long field and LOB data is critical to your business, backing up these table spaces should be considered against the time required to complete the restore task for these table spaces. If the LOB data can be reproduced from a separate source then, when creating or altering a table to include LOB columns, choose the `NOT LOGGED` option.

Note: If you back up a table space that contains table data without the table spaces containing the associated the LONG or LOB fields, you cannot perform point-in-time roll-forward recovery on that table space. All the table spaces that contain any type of data for a table must be rolled forward simultaneously to the same point in time.

While the online backup operation is running, changes can also be occurring on the tables. The roll-forward recovery method is used to ensure that all table changes are captured.

While the general focus of this chapter is on the database, your overall recovery planning should include recovering:

- The operating system and DB2 executables
- Applications, UDFs, and stored procedure code in operating system libraries
- Commands for creating DB2 instances and non-DB2 resources
- Operating system security
- Load copies from a LOAD operation (if you specify COPY YES for the LOAD)

Recovery Time Required

The time required to recover a database is made up of two parts: the time required to complete the restore of the backup; and, if the database is enabled for forward recovery, the time required to apply the logs during the ROLLFORWARD operation. When formulating a recovery plan, you should determine what is a reasonable amount of time for your business operations to be impacted while the database is being recovered.

Testing your overall recovery plan will assist you in determining whether the time required to recover the database is reasonable given your business requirements. Following each test, you may want to increase the frequency with which you take a backup. If roll-forward recovery is part of your strategy, this will reduce the number of logs that are archived between backups and, as a result, reduce the time required to roll forward the database after a restore.

Storage Considerations

When deciding which recovery method to use, consider the storage space required.

The restore method requires space to hold the backup copy of the database and the restored database. The roll-forward method requires space to hold the backup copy of the database or table spaces, the restored database, and the archived database logs.

If a table contains long field or large object (LOB) columns, you should consider placing this data into a separate table space. This will affect your storage space considerations as well as affect your plan for recovery. With a separate table space for long field and LOB data, and knowing the time required to back up long field and LOB data, you may decide to use a recovery plan that only infrequently saves a backup of this long field/LOB table space. You may also choose, when creating or altering a table to include LOB columns, not to log changes to that column. This will reduce the size of the log space required and the corresponding log archive space.

The backup of a SMS table space which contains LOBs can be bigger than the size of the original table space. The backup can be as much as 40 per cent larger depending on the LOB data size in the table space. For example, if you take a backup of a 1GB SMS table space (with LOBs), you will need more than 1GB of disk space when you restore it. This situation only occurs on file systems that support sparse allocation (for example, UNIX operating systems).

To prevent a media failure from destroying a database and your ability to rebuild it, you should keep the database backup, the database logs, and the database itself on different devices. For this reason, it is highly recommended that you use the *newlogpath* configuration parameter to put database logs on a separate device once the database is created. (This and other configuration parameters related to logging are discussed in “Rolling Forward Changes in a Database” on page 216.)

Because the database logs can take a large amount of storage, if you plan on using the roll-forward recovery method you must decide how to manage the archived logs. Your choices include one of the following:

1. Dedicate enough space in the database log path directory to retain the logs.
2. Manually copy the logs to a storage device or directory other than the database log path directory.
3. Use a user exit program to copy these logs to another storage device. (See Appendix J, “User Exit for Database Recovery” on page 753 for more information.)

Note: Under OS/2, the database manager supports a user exit program to handle the storage of both backup copies of databases and database logs on standard and non-standard devices. See Appendix J, “User Exit for Database Recovery” on page 753 for more information.

Keeping Related Data Together

As part of your database design, you will know the relationships that exist between tables. These relationships can be at the application level, where transactions update more than one table, or at the database level, where referential integrity exists between tables, or where triggers on one table affect another table. You should consider these relationships when developing a recovery plan. You will want to back up related sets of data together. The sets of data can be established at either the table space or the database level. By keeping related sets of data together, you can recover to a point where all of the data is consistent. This is especially important if you want to be able to perform point-in-time roll-forward recovery on table spaces.

Recovery Performance Considerations

The following items should be considered when thinking about recovery performance:

- You can improve performance for databases that are frequently updated by placing the logs on a separate device. All database changes are written in the logs.

  In the case of an online transaction processing (OLTP) environment, often more I/O is needed for the logs than to store a data row. Placing the logs on a separate physical disk will minimize disk arm movement that would be required to move between a log and the physical database files.

  You should also consider what other files are on the disk. For example, moving the logs to the same disk used for system paging in a system that has insufficient real memory will defeat your tuning efforts.
- To reduce the amount of time required to complete a restore:
  - Adjust the restore buffer size. The buffer size must be a multiple of the buffer size that was used during the backup.
  - Increase the number of buffers.

    If you use multiple buffers and I/O channels, you should use at least twice as many buffers as channels to ensure that the channels do not have to wait for data. The size of the buffers used will also contribute to the performance of the restore operation. The ideal restore buffer size should be a multiple of the extent size for the table space(s).

    If you have multiple table spaces with different extent sizes, specify a value that is a multiple of the largest extent size.
  - Use multiple target devices.

  - Increase the level of parallelism by using the *intra_parallel* database manager configuration parameter. Increasing the level of parallelism can greatly improve restore performance. For more information about the parameter, see "Enable Intra-Partition Parallelism (intra_parallel)" on page 560.
- If a table contains large amounts of long field and LOB data, restoring it could be very time-consuming. If the database is enabled for forward recovery, the RESTORE command provides the capability to restore selected table spaces. By storing long field and LOB data in separate table spaces, the time required to complete the restore of the data can be reduced by choosing not to restore the table spaces containing the long field and LOB data. If the long field and LOB data is critical to your business, restoring these table spaces should be considered against the time required to complete the back up task for these table spaces. If the LOB data can be reproduced from a separate source then, when creating or altering a table to include LOB columns, choose the `NOT LOGGED` option. If you want to roll forward the table spaces that contain the table, you must roll forward to the end of the logs so that all table spaces that contain the table are consistent.

  Note: If you back up a table space that contains table data without the table spaces containing the associated the LONG or LOB fields, you cannot perform point-in-time roll-forward recovery on that table space. All the table spaces that contain any type of data for a table must be rolled forward simultaneously to the same point in time.

  Recall that long field and LOB data for the same table must be placed in the same table space.
- Offline backups are faster than online backups.
- Multiple I/O buffers and devices should be used.
- Allocate at least twice as many buffers as there are devices being used.
- Do not overload the I/O device controller bandwidth.
- Use more buffers of smaller size rather than a few large buffers.
- Tune the number and the size of the buffers according to the system's resources.

It is also recommended that you monitor and measure within your own system environment. The recommendations are only a starting point: each business and each environment is unique.

Disaster Recovery Considerations

The term **disaster recovery** is used to describe the activities that need to be done to restore the database in the event of a fire, earthquake, vandalism, or other catastrophic events. A plan for disaster recovery can include one or more of the following:

- A site to be used in the event of an emergency
- A different machine on which to recover the database
- Off-site storage of database backups and archived logs

If your plan for disaster recovery is to recover the entire database on another machine, you require at least one full database backup and all the archived logs for the database. When operating your business with this consideration, you may choose to keep a standby database up-to-date by applying the logs to it as they are archived. Or,

you may choose to keep the database backup and log archives in the standby site, and perform a restore/rollforward only after a disaster has occurred. (In this case, a recent database backup is clearly desirable.) With a disaster, however, it is generally not possible to recover all of the transactions up to the time of the disaster.

The usefulness of a table space backup for disaster recovery depends on the scope of the failure. When a major disaster occurs, a full database backup is needed on a standby site. If the disaster is a damaged disk, then a table space backup (for each table space using that disk) can be used to recover. If you have lost access to a container because of a disk failure (or for any other reason), you can restore the container to a different location. For additional information, see “Redefining Table Space Containers During RESTORE” on page 211.

With critical business data being stored in your database, you should plan for the possibility of a natural or man-made disaster affecting your database. Both table space backups and full database backups can have a role to play in any disaster recovery plan. The DB2 facilities available for backing up, restoring, and rolling forward data changes provide a foundation for a disaster recovery plan. You should ensure that you have tested recovery procedures in place to protect your business.

Reducing the Impact of Media Failure

To reduce the possibility of having to recover from a media failure, and to simplify recovering from this type of failure, you should:

- Mirror or duplicate the disks that hold the data and logs for important databases.
- In a partitioned database environment, set up a more rigorous procedure for handling the data and logs on the catalog node. Because this node is very important for maintaining the database, you should put it on a more reliable disk, duplicate it, and take more frequent backups of it. Also try to avoid putting user data on it.

Note: When an I/O error occurs on a table space, the database will “crash.” Following a restart of the database, the table space with the I/O error is disabled while the rest of the database remains accessible.

Protecting Against Disk Failure

If you are concerned about damaged data or active logs due to a disk crash, an area you might wish to consider at some point is the use of some form of tolerance to disk failures. Generally, this would be accomplished through the use of a disk array. A disk array consists of a collection of disk drives that appear as a single large disk drive to an application.

Disk arrays involve disk striping, which is the distribution of a file across multiple disks, mirroring of disks and data parity checks. Through the use of a disk array, the data and logs are protected from disk faults, and you will not lose any transactions which may otherwise happen if disk fault tolerance were not implemented.

Disk arrays are sometimes referred to simply as RAID (Redundant Array of Inexpensive Disks). The specific term RAID generally applies only to hardware disk arrays. Disk arrays can also be provided through software in the operating system or application level. The point of distinction between hardware and software disk arrays is how CPU processing of I/O requests is handled. For hardware disk arrays, disk controllers manage the I/O activity, whereas with software disk arrays this is done by the operating system or application.

Hardware Disk Arrays (RAID)

With a RAID disk array, multiple disks are used and managed by a disk controller, complete with its own CPU. All of the logic required to manage the disks forming the array is contained on the disk controller and so this implementation is operating system independent.

There are five types of RAID architectures, RAID-1 through RAID-5, and each provides disk fault-tolerance. Each of the five has some trade-off in function and performance. By definition, RAID refers to a redundant array. RAID-0, which provides only data striping and not fault-tolerant redundancy, is purposely excluded in this discussion about protecting your data in the event of a disk failure. Although the RAID specification defines five architectures, only RAID-1 and RAID-5 are typically used today.

RAID-1 is also known as disk mirroring or duplexing. Disk mirroring duplicates data (complete file) from one disk onto a second disk using a single disk controller. Disk duplexing is the same as mirroring except disks are attached to a second disk controller (like two SCSI adapters). Data protection is good. Either disk can fail and data is still accessible from the other disk. With duplexing, a disk controller could fail as well and still have complete protection of data. Performance with RAID-1 is also good but the trade-off in this implementation is that the required disk capacity is twice that of the actual amount of data, since data is duplicated on pairs of drives.

RAID-5 involves data and parity striping by sectors. RAID-5 stripes data, sector(s) at a time, across all disks. Parity is interleaved with data information rather than stored on a dedicated drive. Data protection is good. If any disk fails, the data can still be accessed by using the information from the other disks along with the striped parity information. Read performance is good though write performance is considerably worse than that of RAID-1 or normal disk. A RAID-5 configuration requires a minimum of three identical disks. The amount of extra disk space required for overhead varies with the number of disks in the array. In the case of a RAID-5 configuration of 5 disks, the space overhead is 20%.

In using a RAID disk array, a failed disk (except RAID-0) will not prevent users from accessing data on the array. When hot-pluggable or hot-swappable disks are used in the array, a replacement disk can be swapped with the failed disk while the array is in use. For RAID-5, if two disks fail at the same time, all data is lost (but the chance of two disk failures at once is very rare).

You might consider using RAID-1 or software-mirrored disks, described in the next section, for your logs since this provides for recoverability to the point of failure and offers good write performance, which is important for logs. In situations where reliability

is crucial so that time cannot be lost in recovering data in case of a disk failure, and write performance is not quite so critical, consider using RAID-5 disks. Further, if write performance is crucial and you are willing to achieve this with the cost of additional disk space, consider RAID-1 for your data as well as logs.

Software Disk Arrays

A software disk array accomplishes much the same as a hardware disk array but the management of the disk traffic is done by either an operating system task or an application program running on the server. The key point is that like all other programs, the software array must contend for CPU and system resources. This is not a good option for a CPU-constrained system and it should be remembered that overall disk array performance is dependent on the server's CPU load and capacity.

A typical software disk array provides disk-mirroring, as with RAID-1. Although redundant disks are required, a software disk array is comparatively inexpensive to implement since costly RAID disk controllers are not required. One caution with software disk arrays is that having the operating system boot drive in the disk array will prevent your system from starting if that drive fails. If the drive fails before the disk array is running, the disk array cannot start to allow access to the drive. Generally, a boot drive separate from the disk array is also required.

Reducing the Impact of Transaction Failure

To reduce the impact of a transaction failure, try to ensure the following:

- Uninterrupted power supplies.
- Adequate disk space for database logs.
- Reliable communication links among the database partition servers in a partitioned database environment.
- Synchronization of the system clocks in a partitioned database environment. See “System Clock Synchronization in a Partitioned Database System” for more information.

System Clock Synchronization in a Partitioned Database System

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. The time difference among the database partition servers plus any potential operational and communication delays for a transaction should be less than the value found in the Maximum Time Difference Among Nodes (*max_time_diff*) database manager configuration parameter.

To ensure that the log record timestamps reflect the sequence of transactions, DB2 in a partitioned database system uses the system clock on each machine as the basis for the timestamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set backwards, the clock for the logs cannot, and remains at the *same* advanced time until

the system clock exceeds this time. At this time, the log time again reflects the system clock. The implication of this is that a short-term system clock error on a database node can have long-lasting effect on the timestamps of database logs.

As a hypothetical example, assume that the system clock on database partition server A is mistakenly set to November 7, 1999 when the year is 1997, and assume that the mistake is quickly corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point in time between November 7, 1997 and November 7, 1999 is virtually unreachable through roll-forward recovery. When the commit on database partition server A is done the timestamp in the database log is set to 1999, and the clock of the database log stays at November 7, 1999, until the system clock exceeds this time. If you attempt to roll forward to a point in time within the incorrect time frame, the operation will stop at the first timestamp that is beyond the specified stop point, which is November 7, 1997.

Although DB2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the possibility of this type of problem occurring in the database system:

- The configurable values for this parameter range from 1 minute to 24 hours. For information about setting *max_time_diff*, see “Maximum Time Difference Among Nodes (max_time_diff)” on page 560.
- When the first connection request is made to a non-catalog node, this database partition server sends its time to the catalog node for the database. The catalog node then checks that the time on the node requesting the connection and its own time are within the tolerance specified by the *max_time_diff* parameter. If the value specified by the parameter is exceeded, the connection is not allowed.
- An update transaction that involves more than two database partition servers in the database must verify that the time on the participating database partition servers is synchronized before the update can be committed. If two or more database partition servers have a greater time difference than that allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated into other database partition servers.

To correct and prevent an incorrect timestamp in a database log from being propagated further:

1. Adjust the system clock to the correct time.
2. Restore the database partition on the appropriate database partition server with a backup that was taken before the time was incorrectly set.
3. Roll forward the changes to the end of the log for the database partition.
4. Take a back-up copy of the database partition immediately after the changes are rolled forward.

After you do these steps, the log time will be adjusted, the incorrect timestamp will not be propagated, and you will be able to do point-in-time recovery on the database partition from the last backup that you took of the partition.

Recovery Method: Crash Recovery

Crash recovery using the RESTART DATABASE command or the automatic restart enable configuration parameter (*autorestart*) protects a database from being left in an inconsistent, or unusable, state.

The following topics provide additional information:

- Getting to a Consistent Database
- Transaction Failure Recovery in a Partitioned Database Environment

Getting to a Consistent Database

While applications or commands are running against a database, an interruption in power or the failure of an application may cause the immediate cessation or stopping of all activity with the database. One or more of the applications or commands may have started working with the data in the database but not have completed. The partially completed units of work leave the database in an inconsistent, or unusable, state.

In this section, as with each of the recovery methods, planning considerations and how to invoke the specific utilities or commands to carry out the method are reviewed. Then, any concepts or related issues that allow effective use of this method are presented.

See the following topics for more information:

- Planning to Use Automatic Restart
- Enabling Automatic Restart

Planning to Use Automatic Restart

The only consideration is whether you want the rollback of incomplete units of work at the time of a failure to be done by the database manager. If you do, use the automatic restart enable (*autorestart*) configuration parameter. If not, you should be prepared to issue the RESTART DATABASE command when a database failure occurs.

Enabling Automatic Restart

Automatic restart is enabled through the *autorestart* database configuration parameter. The default for this parameter is that automatic restart is “on.” See “Auto Restart Enable (autorestart)” on page 529 for more information.

Transaction Failure Recovery in a Partitioned Database Environment

Database commands and applications can fail for various reasons. A *transaction failure* is not the failure of a database action when it is caused by an incorrect parameter, a limit being exceeded, or a rollback caused by a deadlock. Rather, it is a severe error or condition that causes the database or database manager to end abnormally, and requires that the database be recovered. Examples include events such as a power failure on a machine (causing the database manager and database partitions on it to be down), or a COMMIT/ROLLBACK failure that causes the database to go down because the disk that contains the database log is full, and no additional log files can be allocated for writing the COMMIT/ROLLBACK record.

Typically, database recovery is required on both the failed database partition server and any other database partition server that was participating in the same transaction or application. Database recovery on the failed database partition server is often called *crash recovery*. Crash recovery occurs on the database partition server that failed after the condition that caused the failure is corrected (for example, the power supply is reactivated, or a damaged disk is replaced). Database recovery on the other (still active) database partition servers occurs immediately after the failure is detected. Sometimes called *database partition failure recovery*, in this recovery process, resources are transparently cleaned up for the failed transaction or application.

For more information, see "Failure Recovery on an Active Database Partition Server," and "Transaction Failure Recovery on the Failed Database Partition Server" on page 199 .

Two-Phase Commit Protocol

The discussion of two-phase commit protocol here is to introduce crash recovery in a partitioned database system. For more information about two-phase commit, refer to "Understanding the Two-Phase Commit Process" on page 246.

In a partitioned database environment, the database partition server on which an application is submitted is the coordinator node, and the first agent that works for the application is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. In the first phase, the coordinator node distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

READ-ONLY	No data change occurred at this server
YES	Data change occurred at this server
NO	Because of an error, the server is not prepared to commit

If one of the servers responds "NO," the transaction is rolled back. Otherwise, the coordinator node begins the second phase.

In the second phase, the coordinator node writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded "YES." After all the other database partition servers have committed, they send an acknowledgement of the COMMIT to the coordinator node. The transaction is complete when the coordinator agent has received all COMMIT acknowledgements from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

Failure Recovery on an Active Database Partition Server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator node for an application and the application was running on the failed database partition server (and not

ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, the application receives the SQL error message SQL0279N, and loses its database connection. Otherwise, the coordinator agent will distribute a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.

- If the failed database partition server was the coordinator node for the application, agents that are still working for the application on the active servers are interrupted to do failure recovery. The current transaction is rolled back locally on each server, unless it has been prepared and is waiting for the transaction outcome. In this situation, the transaction is left indoubt. See "Recovering from Problems During Two-Phase Commit" on page 249 for more information about how an indoubt transaction is resolved.
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator node, agents working for this application are interrupted.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

Transaction Failure Recovery on the Failed Database Partition Server

If the failure caused the database manager to end abnormally, when the processor is restarted, you can issue DB2START with the RESTART option to restart the database manager. If you cannot restart the processor, you can also use DB2START to restart the database manager on a different processor. For more information, see the START DATABASE MANAGER command and API in the *Command Reference* and *API Reference* manuals respectively.

An abnormal end may result in database partitions on the server being left in an inconsistent state (meaning that they are unusable). To make them usable, crash recovery is required to make them consistent. Crash recovery can be triggered on a database partition server:

- Explicitly with a RESTART DATABASE command
- Implicitly by a CONNECT request when the *autorestart* database configuration parameter is on.

Crash recovery reapplies the log records in the active log files to ensure that the effect of all complete transactions are in the database. After all the changes are reapplied, all uncommitted transactions are rolled back locally, except for indoubt transactions. In a partitioned database environment, there are two types of indoubt transaction:

- On a database partition server that is not the coordinator node, a transaction is indoubt if it is prepared but not yet committed.
- On the coordinator node, a transaction is indoubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation

occurs when the coordinator agent has not received all the COMMIT acknowledgements from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator node for an application:

- If the server that restarted is not the coordinator node for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.
- If the server that restarted is the coordinator node for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery may not be able to resolve all the indoubt transactions (for example, some of the database partition servers are not available). In this situation, the SQL warning message SQL1061W is returned. You should note that indoubt transactions hold resources, such as locks and active log space. It is possible to get to a point where no changes can be made to the database because the active log space is help up by indoubt transactions. For this reason, you should investigate if indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the LIST INDOUBT TRANSACTIONS command to query, commit, and roll back the indoubt transaction on the server. For more information, see the LIST INDOUBT TRANSACTIONS command and API in the *Command Reference* and *API Reference* manuals respectively.

Note: The LIST INDOUBT TRANSACTIONS command is also used for transactions in a distributed transaction environment. See Chapter 7, “Distributed Databases” on page 239 and Chapter 8, “Using DB2 with an XA-Compliant Transaction Manager” on page 255 for more information about distributed environments. To distinguish between the two types of indoubt transactions, the “originator” field in the output that is returned by LIST INDOUBT TRANSACTIONS displays one of the following:

- DB2 Universal Database Extended Enterprise Edition, which indicates that the transaction originated in the partitioned database environment.
- XA, which indicates that the transaction originated in the distributed environment.

Identifying the Failed Database Partition Server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

SQL0279N This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

SQL1224N This SQLCODE is received when the database partition server that failed is the coordinator node for the transaction.

SQL1229N This SQLCODE is received when the database partition server that failed is not the coordinator node for the transaction.

Determining which database partition server failed is a two-step process. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the `db2nodes.cfg` file.) On the database partition server that detects the error, a message that indicates the node number of the failed server is written in the `db2diag.log` file.

Note: If multiple logical nodes are being used on a processor, the failure of one logical node may cause other logical nodes on the same processor to fail.

Typically, to recover from the failure of a database partition server:

1. Correct the problem that caused the failure.
2. Restart the database manager with the DB2START command from any database partition server.
3. Restart the database with the RESTART DATABASE command on the failed database partition server or servers.

Recovery Method: Restore Recovery

Restore recovery using the BACKUP command in conjunction with the RESTORE command puts the database or table space in a state that has been previously saved.

The following topics provide additional information:

- Backing Up a Database
- Restoring a Database
- Recovery History File Information

Backing Up a Database

To make a backup copy of the database, you use the BACKUP command or the Control Center. Within the Control Center, you select the database to be backed up and then select the backup action.

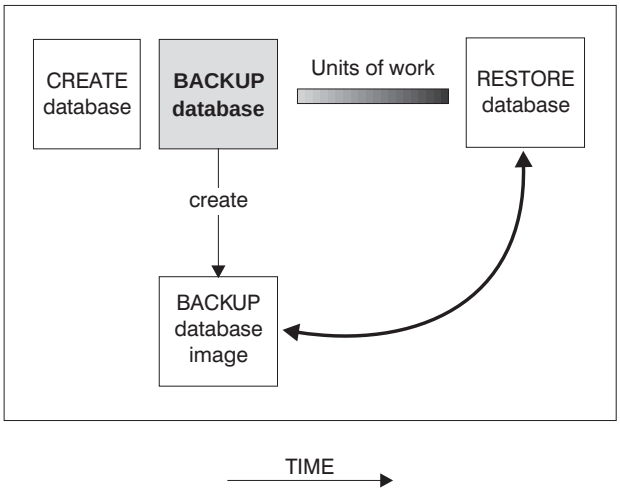


Figure 26. Creating a Database Image

In a partitioned database system, you back up database partitions individually using the BACKUP DATABASE command. The operation is local to the database partition server where you issue the command. You can, however, issue `db2_all` from one of the database partition servers in the instance to submit the backup command on a list of servers, which you identify by their node number. If you do this, you must back up the catalog node first, then back up the other database partitions. You can also use the Control Center to backup database partitions.

In a partitioned database system, you can use the LIST NODES command to determine the list of nodes (database partition servers) that have user tables on them. If you do not require forward recovery capability, regularly back up the database on this list of nodes. If you want to be able to do forward recovery, you must regularly back up the database on the list of nodes, and you must have at least one backup of the rest of the nodes in the system (even those that do not contain user data for that database).

Two situations require the backed-up image of a database partition at a database partition server that does not contain user data for the database:

- You added a database partition server to the database system after taking the last backup, and you need to do forward recovery on this database partition server.
- Point-in-time recovery is used, which requires that all database partitions in the system are in the roll-forward pending state.

You must keep in mind the recovery method to be used. The following sections provides requirements and other considerations that apply to this task:

- Planning to Use the BACKUP Command
- Invoking the BACKUP Command
- Backup Images Created by BACKUP

Planning to Use the BACKUP Command

Your planning considerations should include:

- You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the BACKUP command.
- The database may be local or remote. The backup remains on the database server unless a storage management product such as ADSTAR* Distributed Storage Manager (ADSM) is used.
- You can back up a database or table space to a fixed disk, a tape, or a location managed by ADSM or another vendor storage management product. See "ADSTAR Distributed Storage Manager" on page 230 for information on ADSM.

  Under OS/2, you can also back up to diskette or to a user exit.
- Under Windows NT and Windows 95, you can back up to diskette.
- Under OS/2, a user exit is used when backing up to tape because the operating system has no native tape support.

  Under UNIX-based operating systems and Windows NT, native tape support is available.
- Roll-forward recovery is not enabled by the default setting ("Off") of the *logretain* and *userexit* configuration parameters. The default for both parameters is set to "Off" because, initially, there is no backup that you can use to recover the database; initially, the database cannot be recovered, so you cannot perform forward recovery on it.

  To enable a new database for roll-forward recovery, you must enable at least one of these configuration parameters before taking the first backup of the database. When you change the value of one or both parameters, the database will be put into the *backup pending* state, which requires that you take an offline backup of the database. After the backup operation completes successfully, the database can be used.
- A table space backup and a table space restore cannot be run at the same time, even if the backup and restore are working on different table spaces.
- In OS/2, when backing up a database online to a user exit, please note that the database will be quiesced before the backup starts. As such, the backup will wait for all transactions to either commit or rollback before it starts. While the backup is running, all new transactions will wait until the backup is complete, and, once the backup is completed, all transactions will continue processing as usual.
- If you have tables that span more than one table space, you should backup (and restore) the set of table spaces together.
- If your database is enabled for roll-forward recovery and you are using a tape system that does not support the ability to uniquely reference a backup, it is recommended that you do not keep multiple backup copies of the same database on the same tape.
- Multiple files may be created to contain the backed up data from the database or table space.

In OS/2, when you restore from a user exit and roll forward the database, the path to the database is the only reference used to locate the containers. Therefore, all the containers for that database that are on the backup tape are restored.

- In a partitioned database environment, an offline backup uses an exclusive connection to the database at that database partition server (that is, the operation requires an exclusive connection to the database partition), so no other application can be connected to the database partition. When you do an offline backup of the catalog node, there can be no activity on the *entire* database, including backups of the database on non-catalog database partition servers. You can use `db2_all` to back up the database, but you must ensure that the catalog node is backed up first. After the catalog node is backed up, the other database partitions can be backed up at the same time.
- In a partitioned database system, you should also keep a copy of the `db2nodes.cfg` file with any backup copies you take, as protection against possible damage to this file.

Invoking the BACKUP Command

The following considerations are useful when running the BACKUP command:

- You must start the database manager (DB2START) before running the BACKUP command or API. When using the Control Center, you do not need to explicitly start the database manager.
- When using the command, API, or task under Control Center, you must specify a database alias name, not the database name itself.
- To reduce the amount of time required to complete a backup:
  - Use table space backups.
  - Increase the backup buffer size.
  - Increase the number of buffers.

    If you use multiple buffers and I/O channels, you should use at least twice as many buffers as channels to ensure that the channels do not have to wait for data. The size of the buffers used will also contribute to the performance of the backup operation. The ideal backup buffer size should be a multiple of the extent size for the table space(s).

    If you have multiple table spaces with different extent sizes, specify a value that is a multiple of the largest extent size.

    You may specify the number of pages to use for each backup buffer when you invoke the BACKUP command. The minimum number of pages is 16. If you do not specify the number of pages, each buffer will be allocated based on the database manager configuration parameter *backbufsz*. If there is not enough memory available to allocate the buffer, an error will be returned.

    For details about the configuration parameter, see “Default Backup Buffer Size (backbufsz)” on page 475.
  - Use multiple target devices.

  - Increase the level of parallelism by using the *intra_parallel* database manager configuration parameter. For more information about this parameter, see "Enable Intra-Partition Parallelism (intra_parallel)" on page 560.

- You may select to only carry out a backup for part of a database (and subsequent recovery) by using the TABLESPACE option of the BACKUP command. This makes administering data, index, and long fields/large objects (LOBs) in separate table spaces easier.
- In OS/2, when backing up a database to removable media, such as tape, the database manager writes information to media volume 1. Once the first media is in the drive, do not remove the media unless the operating system backup facility prompts you for media 2.
- If each table space is on a different disk, a media error only affects a particular table space, not the entire database. The table space with the error is placed in a roll-forward pending state. You are still able to use the other table spaces in the database.

  If the table space in this state has the system catalog tables, you are not able to connect to the database.
- The system catalog table space can be restored independent of the rest of the database if a table-space level backup containing the system catalog table space is available.
- The backup will fail if a list of the table spaces to be backed up contains a temporary table space.
- You cannot back up a database that is not in a usable state except for a database in the backup pending state.
  - If a database or a table space is in a partially restored state due to a system crash during any stage of restoring the database, you must successfully restore the database or the table space before you can back it up.
  - After a restore, a database configured for roll-forward recovery might be in a roll-forward pending state. If this is the case, you must roll it forward before it is usable.
  - If a database was created with a previous release of the database manager and the database has not been migrated, you must migrate the database before you can back it up.

    See Appendix A, "Planning Database Migration" on page 579, for information about migrating a database.
  - If any of the table spaces in a database is in an "abnormal" state, you cannot back up the database.
- If a system crash occurs during a critical stage of backing up a database, you cannot successfully connect to the database until you re-issue the BACKUP command.

- The BACKUP command provides a concurrency control for multiple processes that are making backup copies of different databases. The control keeps the backup target device open until the entire backup process has ended.

  If an error occurs during a backup process and the open container cannot be closed, other backup processes to the same target drive may receive access errors. To correct any access errors, you must completely exit the backup process that caused the error and disconnect from the target device.
- If you are using the BACKUP command for concurrent backup processes to tape, ensure that the processes do not target the same tape.

Backup Images Created by BACKUP

Backup images are created at the target specified when you call the BACKUP command:

- In the directory for disk or diskette backups
- At the device specified for tape backups
- At an ADSTAR Distributed Storage Manager (ADSM) server
- At another vendor's server
- For OS/2, through the use of a user exit

The recovery history file is updated automatically with summary information whenever you carry out a backup or restore of a full database or table space. This file can be a useful tracking mechanism for restore activity within a database. This file is created in the same directory as the database configuration file. For more information on the recovery history file, see “Recovery History File Information” on page 214.

In UNIX-based environments, the file name(s) created on disk will consist of a concatenation of the following information, separated by periods; on other platforms a four-level subdirectory tree is used:

Database alias
A 1-to-8 character database alias name that was supplied when the backup command was invoked.

Type
Type of backup taken, where: “0” is for full database, “3” is for table space, and “4” is for copy from a table load.

Instance name
A 1-to-8 character name of the current instance of the database manager that is taken from the `DB2INSTANCE` environment variable.

Node number
The node number.

Catalog node number
The node number of the database's catalog node.

Time stamp
A 14-character representation of the date and time the backup was performed. The timestamp is in the format *yyyymmddhhnnss*, where:

yyyy is the year (1995 to 9999)
mm is the month (01 to 12)
dd is the day of the month (01 to 31)

hh is the hour (00 to 23)
nn is the minutes (00 to 59)
ss is the seconds (00 to 59)

Sequence number A 3-digit sequence number used as a file extension.

In UNIX-based operating systems, the format would appear as:

```
Database alias.Type.Instance name.node nnnn.catn nnnn.timestamp.number
```

On other operating systems, the format would appear as:

```
Database alias.Type\Instance name.node nnn\catn nnn\yyyymmdd\hhmmss.number
```

For example in UNIX-based environments, a database named STAFF on the DB201 instance may be backed up on disk to a file named:

```
STAFF.0.DB201.NODE0000.CATN0000.19950922120112.001
```

For tape-directed output, file names are not created; however, the above information is stored in the backup header for later verification purposes.

Notes:

1. If you want to use tape media for database back-up and restore operations, a tape device must be available through the standard operating system interface. On a large partitioned database system, however, it may not be practical to have a tape device dedicated to each database partition server. You can connect the tape devices to one or more ADSM servers, so that access to these tape devices is provided to each database partition server.
2. On a partitioned database system, you can also use products that provide virtual tape device functions, such as REELlibrarian 4.2 or CLIO/S. You use these products to access the tape device connected to other nodes (database partition servers) through a pseudo tape device. Access to the remote tape device is provided transparently, and the pseudo tape device can be accessed through the standard operating system interface.

Restoring a Database

The following sections provide requirements and other considerations that apply to the RESTORE command:

- Planning to Use the RESTORE Command
- Invoking the RESTORE Command
- Redefining Table Space Containers During RESTORE
- Restoring to an Existing Database
- Restoring to a New Database

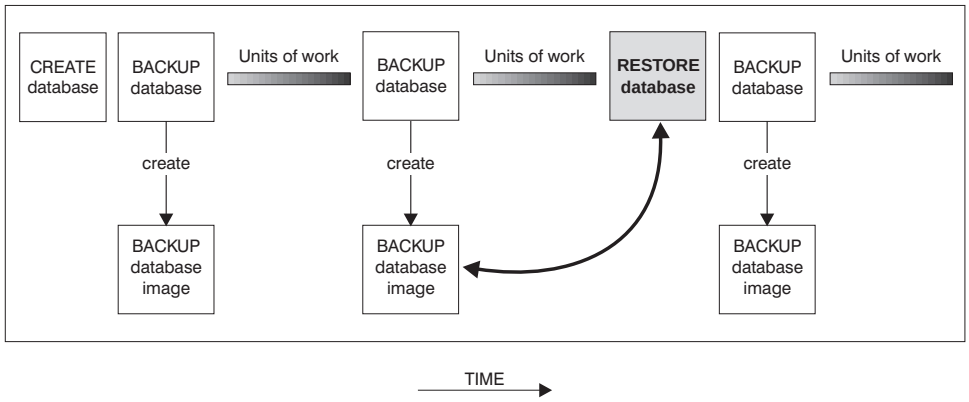


Figure 27. Restoring a Database Using a Backup Image

Note: Because restore is required as part of the roll-forward recovery method, all information about restore also applies to that method as well.

Planning to Use the RESTORE Command

You should consider the following:

- You must have SYSADM, SYSCTRL, or SYSMAINT, authority to restore to an existing database from a full database, or table space, backup. To restore to a new database, you must have SYSADM or SYSCTRL authority.
- You can only use this command if the database or table space has been previously backed up with the BACKUP command.
- If you use the Control Center, you cannot restore backups that were taken previous to DB2 Version 5.
- In OS/2, the RESTORE command can call a user exit program only if a user exit program was used to backup the database.
- If you backed up tables that spanned more than one table space, you should restore the set of table spaces together.
- You can choose at the time of the restore which type of restore is to be carried out. You can select from the following types:
  - A full restore of everything from the backup
  - A table space restore
  - A restore of only the recovery history file
- When doing a partial or subset RESTORE, you can use either a table space backup, or a full database backup and choose one or more table spaces from that image. All the log files associated with the table space (or table spaces) must exist from the time the backup was created.

  In a partitioned database system, if you intend to roll forward a table space (or table spaces) to the end of the logs, you do not have to restore it at each database

partition (node). You only need to restore it at the database partitions that require recovery. If you intend to roll forward a table space to a point in time, you must restore the table space at each database partition before rolling forward.

- You can perform a partial or subset restore of a backup created using Version 5 of DB2. This cannot be done with earlier versions of DB2. A partial or subset restore is only possible with a recoverable database. A recoverable database has either the *logretain* or *userexit* (or both) configuration parameters enabled to allow roll-forward recovery to occur. With a recoverable database, once the restore is complete, the table space (or table spaces) must be rolled forward (at the end of the restore, each table space that was restored is in the roll-forward pending state).
- In OS/2, a partial or subset restore is not possible when restoring from a user exit.
- The RESTORE command can use the ADSTAR Distributed Storage Manager (ADSM) utility, and any restrictions of that utility should also be considered. (See "ADSTAR Distributed Storage Manager" on page 230.)
- Another vendor storage management product may also be used if that product was used to store the original backup.
- A database restore requires an exclusive connection: that is, no applications can be running against the database when the task is started. Once it starts, it prevents other applications from accessing the database until the restore is completed.
- A table space restore can be online (share mode) or offline (exclusive mode).

  Note: You cannot do an online table space restore of the system catalog tables.
- The database may be local or remote.

Invoking the RESTORE Command

The following considerations are useful when running the RESTORE command:

- The database manager must be started before restoring a database.
- The database to which you restore the data may be the same one as the data was originally backed up from, or it may be different. You may restore the data to a new or an existing database.

  Note: If you are restoring a table space (or table spaces), the backup image must have been taken from the same database that holds the table space (or table spaces).
- A database enabled for roll-forward recovery must be rolled forward after it is restored, unless a restore without roll-forward is specified through the RESTORE command. You may not turn roll-forward off if the backup is taken online or if the backup is taken at the table space level.
- During the restore procedures, you have the ability to optionally select to use multiple buffers to improve the performance of the restore procedure. The multiple internal buffers may be filled with data from the backup media.

  You may specify the number of pages to use for each restore buffer when you invoke the RESTORE command. The value you specify must be a multiple of the number of pages that you specified for the backup buffer. The minimum number of

pages is 16. If you do not specify the number of pages, each buffer will be allocated based on the database manager configuration parameter *restbufsz*. If there is not enough memory available to allocate the buffer, an error will be returned.

For details about the configuration parameter, see “Default Restore Buffer Size (restbufsz)” on page 476.

- The TAKEN AT parameter of the RESTORE DATABASE command requires the timestamp for the backup. The timestamp can be exactly as it was displayed after the completion of a successful BACKUP command, that is in the format yyyymmddhhmmss.

  You can also specify a partial timestamp. For example, assume that you have two different backups with the timestamps 19971001010101 and 19971002010101. If you specify 19971002 for TAKEN AT, the 19971002010101 backup is used.

  If TAKEN AT is not specified, there must only be one backup on the source media.
- The backup copy of the database or table space to be used by the RESTORE command can be located on a fixed disk, a tape or a location managed by the ADSTAR* Distributed Storage Manager (ADSM) utility or another vendor storage management product. See “ADSTAR Distributed Storage Manager” on page 230 for information on ADSM.

  Under OS/2, the backup copy of the database or table space could also be located on diskette or through a user exit.

  Under Windows 95 and Windows NT, the backup copy of the database or table space could also be located on diskette.
- While restore and roll-forward are independent operations, your recovery strategy may have restore as the first phase of a complete roll-forward recovery of a database. After a successful restore, a database that was configured for roll-forward recovery at the time the backup was taken enters a roll-forward pending state, and is not usable until the ROLLFORWARD command has been run successfully.

  When the ROLLFORWARD command is issued:
  - If the database is in the roll-forward pending state, the database is rolled forward.
  - If the database is not in the roll-forward pending state, but table spaces in the database are, when you issue the ROLLFORWARD command and specify a list of table spaces, only those table spaces are rolled forward. If you do not specify a list, all table spaces that are in the roll-forward pending state are rolled forward.

  Another database RESTORE is not allowed when the roll-forward process is running.

Notes:

1. If you are restoring from a full database backup that was created using the **offline** option of the BACKUP command, you can bypass this roll-forward pending state during the restore process. Using the WITHOUT ROLLING FORWARD option allows you to use the restored database immediately without rolling forward the database.
2. If you are restoring from a backup that was created using the **online** option of the BACKUP command, you **cannot** bypass this roll-forward pending state.

- Once the RESTORE command starts, the database is not usable until the RESTORE command completes successfully.
- Once the RESTORE command starts for a table space backup, the table space is not usable until the RESTORE command followed by a roll-forward recovery completes successfully.
- If a system failure occurs during any stage of restoring a database, you cannot connect to the database until you reuse the RESTORE command and successfully complete the restore.
- If a system failure occurs during the restoring of a table space backup, only the table space being restored is unusable. The other table spaces in the database can still be used.
- If the code page of the database being restored does not match a code page available to an application; or, if the database manager does not support code page conversions from the database code page to a code page that is available to an application; then the restored database will not be usable.
- In OS/2, if you backed up your database using the *sqluback* API in a previous release of DB2, then you must use the *sqludres* API to restore your database. However, this API is no longer supported by the command line. To restore a back-level backup from the command line, use the *db2resdb* utility provided in the `misc` subdirectory of the `sqllib` directory. This utility will make the call to the *sqludres* API on your behalf, restore the database to the target drive, then attempt to migrate it to the current release.

  The syntax for this utility is:

  ```
  db2resdb <dbname> <source drive> <target drive>
  ```

  where

  ```
  dbname = The name of the database which was backed up
  source drive = The drive letter where the backup resides
  target drive = The drive letter where the database is to be created
  ```

Redefining Table Space Containers During RESTORE

During a backup of a database or one or more table spaces, a record is kept of all the table space containers in use by the table spaces that are backed up. During a RESTORE, all containers listed in the backup are checked to see if they currently exist and are accessible. If one or more of the containers is inaccessible because of a media failure (or for any other reason), the RESTORE will fail. In order to allow a restore in such a case, the **redirecting** of table space containers is supported during the

RESTORE. This support includes adding, changing, or removing of table space containers.

There are cases in which you want to restore even though the containers listed in the backup do not exist on the system. An example of such a case is where you wish to recover from a disaster on a system other than that from which the backup was taken. The new system may not have the necessary containers defined. In order to allow a RESTORE in this case, the **redirecting** of table space containers at the time of the RESTORE to alternate containers is supported.

In both situations, this type of RESTORE is commonly referred to as a **redirected restore**.

You can redefine table space containers through the restore task from within the Control Center. You can also use the REDIRECT parameter of the RESTORE command to specify the redirection. If you are using the Control Center, one way of performing a redirected restore is to use the Containers page of the Restore Database notebook. This page provides function that you can use to add new containers, change the path of an existing container, or remove a container. If, during the process of the restore database operation an invalid container path is detected, the Control Center will prompt you to either change the container path, or remove the container.

Roll-forward Recovery Considerations: A RESTORE is often followed by a ROLLFORWARD to reapply changes recorded in the database logs after the point in time where the backup was taken. During a roll-forward operation, you may re-execute or re-run a transaction which carries out an ALTER TABLESPACE with the `ADD` option (to add a container). For the ROLLFORWARD to be successful, the container to be added must be accessible. If the container is not accessible, then the roll-forward for the table space is suspended, and the table space is left in a roll-forward pending state.

You may or may not wish to re-do the add container operations in the database logs. Recalling our previous disaster recovery example, you may not know which containers may have been added since the backup was taken. Therefore, you cannot anticipate which containers are needed. Alternatively, depending on why you are performing a redirected restore, you may simply prefer the list of containers you specified at the time of the restore, and do not want any other containers added. To control this behavior, you can indicate at the time of the restore whether you want the ROLLFORWARD to re-create the containers during the roll-forward recovery. This is specified on the `CONTAINERS - CHANGE` window where you edit the list of table space containers.

Notes:

1. Directory and file containers are automatically created if they do not exist. No redirection is necessary unless the containers are inaccessible for some other reason. The database manager does not automatically create device containers.
2. The ability to perform container redirection on any RESTORE provides considerable flexibility in managing table space containers. For example, even though we do not directly support adding containers to SMS table spaces, you could accomplish this by simply specifying an additional container on a redirected

restore. Similarly, you could move a DMS table space from file containers to device containers.

3. Redirected restore is also supported through a number of APIs. Although you could write a program to perform redirected restore for a specific case, these APIs are primarily intended for developers who want to produce a general purpose utility.

Restoring to an Existing Database

You may restore a backup copy of a full database backup or table space backup to an existing database or one or more table spaces. To restore to an existing database, you must have SYSADM, SYSCTRL, or SYSMAINT authority. The backup image may differ from the existing database in its alias name, its database name, or its database seed.

A database seed is a unique identifier of a database that remains constant for the life of the database. This seed is assigned by the database manager when the database is first created. The seed is unchanged following a restore of a backup even if the backup has a different database seed. DB2 always uses the seed from the backup.

Note: You can only restore a table space if the table space currently exists, and it is the same table space. (The “same table space” means that the table space was not dropped and re-created between taking the backup image and the attempt to restore the table space.)

When restoring to an existing database, the restore task performs the following functions:

- Delete table, index, and long field contents for the existing database, and replace them with the contents from the backup.
- Replace table space table entries for each table space being restored.
- Retain recovery history file unless the one on disk is damaged. If the file on the disk is damaged, the database manager will copy the file from the backup.
- Retain the authentication for the existing database.
- Retain the database directories for the existing database that define where the database resides and how it is cataloged.
- When the database seeds are different:
  - Delete the logs associated with the existing database
  - Copy the database configuration file from the backup
  - Change the database configuration file to indicate that the default log file path should be used for logging
- When the database seeds are the same:
  - Retain the current database configuration file, unless the file is corrupted, in which case this file will be copied from the backup.
  - Delete the logs if the image is of a non-recoverable database; otherwise, the logs will be kept. The log path (which is specified by the *logpath* parameter) is also changed to the value specified in the database configuration file that is in the backup.

Restoring to a New Database

As an alternative to restoring a database to a database that already exists, you may create a new database and then restore the backup of the data. To restore to a new database, you must have SYSADM or SYSCTRL authority.

Notes:

1. You cannot restore a table space backup to a new database.
2. The code pages of the backup and the target database must match. If they do not, first create the new database specifying the correct code page, then restore it.

In this case, the RESTORE command will perform the following functions:

- Create a new database, using the database name and database alias name that was specified by the target database alias parameter. (If this target database alias was not specified, the RESTORE command will create a database with the name and alias the same as the source database alias parameter.)
- Restore the database configuration file from the backup.
- Modify the database configuration file to indicate that the default log file path should be used for logging.
- Restore the authentication type from the backup.
- Restore the database comments from the backup for the database directories.
- Restore the recovery history file for the database.

Recovery History File Information

A recovery history file is created with each database and is automatically updated whenever there is a:

- Database or table space backup
- Database or table space restore
- Database or table space roll forward
- Table space quiesce
- Load of a table

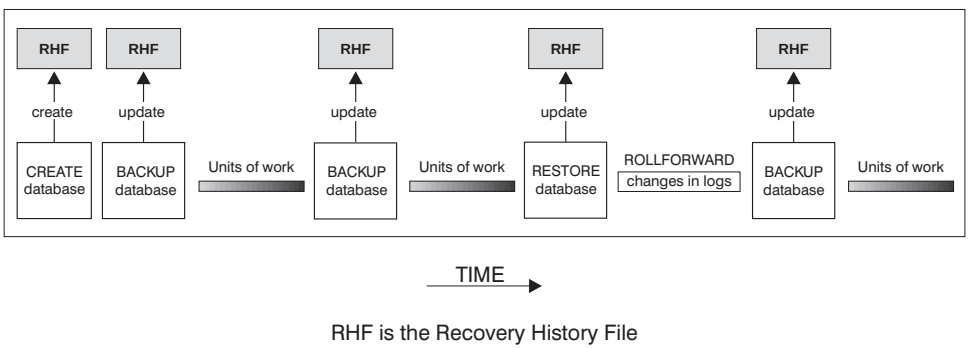


Figure 28. Creating and Updating the Recovery History File

The file contains a summary of the backup information that can be used in the event that **all** or **part** of the database must be recovered to a given point in time. The information in the file includes:

- The part of the database that was copied and how
- The time the copy was made
- The location of the copy (stating both the device information and the logical way to access the copy)
- The last time a restore was done.

Every backup operation (both table space and full database) includes a copy of the recovery history file. The recovery history file is linked to the database. Dropping a database deletes the recovery history file. Restoring a database to a new location restores the recovery history file. Restoring does not overwrite the existing history recovery file.

If the current database is unusable or not available and the associated recovery history file is damaged or deleted, an option on the RESTORE command allows only the recovery history file to be restored. The recovery history file can then be reviewed to provide information on which backup to use to restore the database.

The size of the file is controlled by the *rec_his_retentn* configuration parameter (see "Recovery History Retention Period (rec_his_retentn)" on page 531) that specifies a retention period (in days) for the entries in the file. Even if the number for this parameter is set to zero (0), the most recent full database backup plus its restore set is kept. (The only way to remove this copy is to use the PRUNE with FORCE option.) The retention period has a default of 366 days. The period can be set to an indefinite number of days by using -1. In this case, explicit pruning of the file is required.

You can query and run commands against the recovery history file by using an API function call, the command line processor, or the Control Center. The five (5) basic queries and commands are: OPEN, CLOSE, GET NEXT, UPDATE, and PRUNE. (For more information on the command syntax see the *Command Reference*. For more information on the API function call, see the *API Reference*. For more information on the Control Center, access the Control Center from your workstation.)

Recovery Method: Roll-Forward Recovery

When you first create a database, only circular logging is enabled for it. This means that logs are re-used (in a circular fashion), and are not saved or archived. With circular logging, roll-forward recovery is not possible: only crash recovery or a restore of the database to the time of the last backup is enabled. When log archiving is performed, however, roll-forward recovery is possible, because the logs record changes to the database after the time that the backup was taken. You perform log archiving by activating either (or both) of the *logretain* and *userexit* database configuration parameters. When either of these parameters are enabled, the database is enabled for *roll-forward recovery*.

Roll-forward recovery re-applies the completed units of work recorded in the logs to the restored database, table space, or table spaces. You can specify that roll-forward recovery is to the end of the logs, or to a particular point in time.

Roll-forward recovery can follow the completion of a full database restore as described in “Restoring a Database” on page 207. It can also be done with table spaces that are in a roll-forward pending state. For considerations on rolling forward a table space, see “Rolling Forward Changes in a Table Space” on page 220

For more information about the database configuration parameters associated with logging, see “Configuration Parameters for Database Logging” on page 217.

Rolling Forward Changes in a Database

Roll-forward recovery builds on a restored database and allows you to restore a database to a particular time that is after the time that the database backup was taken. This point can be either the end of the logs, or a point between the time of the database backup and the end of the logs.

You might use point-in-time recovery if an active or an archived log is not available. In this situation, you could roll forward to the point where the log is missing. You might also roll forward to a point in time if a bad transaction was run against the database. In this situation, you would restore the database, then roll forward to just before the time that the bad transaction was run.

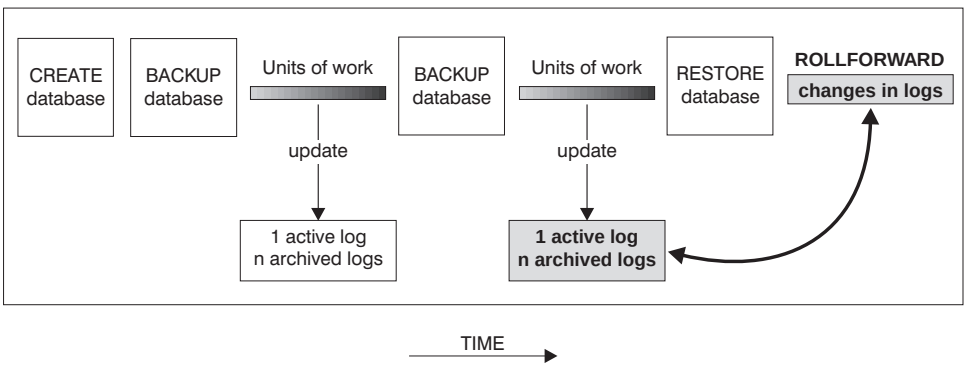


Figure 29. Roll-Forward Recovery

You can also perform point-in-time roll-forward recovery on table spaces. For additional information, see “Rolling Forward Changes in a Table Space” on page 220.

To use this method, the database must be configured to enable roll-forward recovery. Considerations for the database configuration file and database logs are presented in the following topics:

- Configuration Parameters for Database Logging
- Rolling Forward Changes in a Table Space

- Planning to Use the ROLLFORWARD Command
- Invoking the ROLLFORWARD Command
- Using the Load Copy Location File
- Considerations for Managing Log Files
- Losing Logs

Configuration Parameters for Database Logging

The database configuration file contains parameters related to roll-forward recovery. The default parameters do not support this recovery, so if you plan to use it, you need to change some of these defaults. For additional information, see Chapter 19, "Configuring DB2" on page 459.

Primary logs (logprimary)

This parameter specifies the number of primary logs that will be created.

A primary log, whether empty or full, requires the same amount of disk space. Thus, if you configure more logs than you need, you use disk space unnecessarily. If you configure too few logs, you can encounter a log-full condition. As you select the number of logs to configure, you must consider the size you make each log and whether your application can handle a log-full condition.

If you are enabling an existing database for roll-forward recovery, change the number of primary logs to the sum of the number of primary and secondary logs, plus 1. Additional information is logged for **long varchar** and **LOB** fields in a database enabled for roll-forward recovery.

Secondary logs (logsecond)

This parameter specifies the number of secondary log files that are created and used for recovery log files (only as needed).

When the primary log files become full, the secondary log files (of size *logfilsiz*) are allocated one at a time as needed, up to a maximum number as controlled by this parameter. An error code will be returned to the application, and activity against the database will be stopped, if more secondary log files are required than are allowed by this parameter.

See "Number of Secondary Log Files (logsecond)" on page 522 for recommendations on how to use secondary logs.

Log size (logfilsiz)

This parameter determines the number of pages for each of the configured logs. A page is 4KB in size.

The size of each primary log has a direct bearing on performance. When the database is configured to retain logs, each time a log is filled, a request is issued for allocation and initialization of a new log. Increasing the size of the log reduces the number of requests required to allocate and initialize new logs. (Keep in mind, however, that with a larger log size it takes more time to format each new log). The formatting of new logs is transparent to applications connected to the database so that database performance is unaffected by formatting.

Assuming that you have an application that keeps the database open to minimize the processing time to open a database (see “Recovery Performance Considerations” on page 191), the value for the log size should be determined by the amount of time it takes to make offline archived log copies.

The data transfer speed of the device you use to store offline archived logs, and the software used to make the copies, must at a minimum match the average rate at which the database manager writes data in the logs. If the transfer speed cannot keep up with new log data being generated, you may run out of disk space if logging activity continues for a sufficiently long period of time, determined by the amount of free disk space. If this happens, database processing will stop.

The data transfer speed is most significant when using tape or some optical medium. (Refer to Appendix J, “User Exit for Database Recovery” on page 753 for information on using different media for storing logs.) Some tape devices require the same amount of time to copy a file, regardless of its size. You must determine the capability of your archiving device.

Additionally, tape devices have some unique considerations. The frequency of the archiving request is important. If the time for any copy operation is five minutes, the log size should be large enough to hold five minutes of log data during your peak work load. Also, the tape device may have design limits that restrict the number of operations per day. These factors must be considered when you determine the log size.

Minimizing log file loss is also an important consideration in setting the log size. Archiving takes an entire log. If you use a single large log, you increase the time between archiving. If the medium containing the log fails, some transaction information will probably be lost. Decreasing the log size increases the frequency of archiving but can reduce the amount of information loss in case of a media failure since the smaller logs before the one lost can be used.

Log Buffer (logbufsz)

This parameter allows you to specify the amount of database shared memory to use as a buffer for log records before writing these records to disk. The log records are written to disk when one of the following occurs:

- A transaction commits
- The log buffer is full
- As a result of some other internal database manager event.

Buffering the log records will result in more efficient logging file I/O, because the log records will be written to disk less frequently and more log records will be written at each time.

Number of Commits to Group (mincommit)

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed. This delay can help reduce the database manager overhead associated with writing log records

and, as a result, improve performance when you have multiple applications running against a database and many commits are requested by the applications within a very short time frame.

This grouping of commits will only occur when the value of this parameter is greater than 1, and when the number of applications connected to the database is greater than the value of this parameter. When commit grouping is being performed, application commit requests are held until the earlier of either one second elapsing or the number of commit requests equals the value of this parameter.

New log path (newlogpath)

A database log is initially created in SQLOGDIR, which is a subdirectory of the database directory. You may change the location where active logs and future archive logs are placed by changing this configuration parameter. Archive logs stored in the database log path directory are not moved to the new log path directory if the database is configured for roll-forward recovery.

Because you can change the log path directory, the logs needed for roll-forward recovery may exist in different directories. You can change this configuration parameter during the roll-forward process to allow you to access logs in multiple directories.

The directory path change will not be applied unless the database is in a consistent state. A database configuration parameter indicates the status of the database. See “Database is Consistent (database_consistent)” on page 539 for additional information about this status indicator. See “Considerations for Managing Log Files” on page 227 for information about the roles database logs play if a database is not in a consistent state.

Log retain (logretain)

This parameter causes archived logs to be kept in the database log path directory. Enabling it allows the database manager to use the roll-forward recovery method. You do not require *userexit* to be enabled when the *logretain* configuration parameter is enabled. Either one of the two parameters is sufficient to allow the roll-forward recovery method.

Using this parameter means that the circular logging, that is the default, is being overridden.

User exit (userexit)

This parameter causes the database manager to call a user exit program for archiving and retrieving logs. With the user exit enabled, roll-forward recovery is allowed. You do not require *logretain* to be enabled when the *userexit* configuration parameter is enabled. Either one of the two parameters is sufficient to allow the roll-forward recovery method.

Using this parameter means that the circular logging, that is the default, is being overridden. *Userexit* implies *logretain* but the reverse is not true.

See Appendix J, “User Exit for Database Recovery” on page 753, for information about the user exit program.

The active log path is important when using either the *userexit* configuration parameter or the *logretain* configuration parameter to allow roll-forward recovery. When the *userexit* configuration parameter is set, the user exit is called to archive log files away from the active log path. When the *logretain* configuration parameter is set, this ensures that the log files remain in the active log path. The active log path is determined either by the Path to Log Files or Changed Path to Log Files (*newlogpath*).

Rolling Forward Changes in a Table Space

If the database is enabled for forward recovery, you have the option of backing up, restoring, and rolling forward table spaces independent of the database. You may want to implement a recovery strategy for individual table spaces because this can save time: it takes less time to recover a portion of the database than it does to recover the entire database. For example, if a disk is bad and it only contains one table space, the table space can be restored and rolled forward without having to recover the entire database (and without impacting user access to the rest of the database). Also, table-space-level backups allow you to back up critical portions of the database more frequently than other portions, which requires less time than backing up the entire database.

You can issue QUIESCE TABLESPACES FOR TABLE to create a transaction-consistent point in time that you can use for rolling forward table spaces. When you quiesce table spaces for a table (in share, intent to update or exclusive), the request will wait (through locking) for all running transactions that are accessing objects in the table spaces to complete while blocking new requests against the table spaces. When the quiesce request is granted, all outstanding transactions are already completed (committed or rolled back) and the table spaces are in a consistent state. You can look in the recovery history file to find quiesce points and check whether they are past the minimum roll-forward time to determine a desirable time for a ROLLFORWARD STOP.

Different states are associated with a table space to indicate its current status:

- A table space will be placed in the *roll-forward pending* state after it is restored, or following an I/O error. When the I/O error is corrected, the table space must be rolled forward to remove the roll-forward pending state. If the table space has been restored, it must be rolled forward.
- A table space will be placed in the *roll-forward-in-progress* state when a roll-forward operation is in progress on that table space. The table space will be removed from the roll-forward-in-progress state when ROLLFORWARD completes successfully.
- A table space will be placed in the *restore pending* state after a ROLLFORWARD CANCEL or a ROLLFORWARD in which an unrecoverable error occurs on that table space. The table space must be restored and rolled forward again.
- A table space will be placed in the *backup pending* state after a ROLLFORWARD to a point in time, or after a LOAD NO COPY operation. The table space must be backed up.

If the data and long objects of a table are in separate table spaces, and the table has been reorganized, the table spaces for both the data and long objects must be restored

and rolled forward together. You should take a back up of the affected table spaces after the table is reorganized.

After a table space is restored, it is always in the roll-forward pending state (that is, if you restore a table space and specify the WITHOUT ROLLING FORWARD parameter, the WITHOUT ROLLING FORWARD is ignored). To make the table space usable, you must perform roll-forward recovery on it. You have the option of rolling forward to the end of the logs, or rolling forward to a point in time. If you want to roll forward a table space to a point in time, you should be aware of the following:

- You cannot roll forward system catalog tables to a point in time. These must be rolled forward to the end of the logs to ensure that all table spaces in the database remain consistent.
- A table space that is to be rolled forward to a point in time must have been restored from a backup that is earlier than the point in time specified for the roll forward.
- If a table is contained in multiple table spaces, all table spaces that contain the table must be rolled forward simultaneously. If, for example, the table data is contained in one table space, and the index for the table is contained in another table space, you must roll forward both table spaces simultaneously to the same point in time.
- Before rolling forward a table space, use the LIST TABLESPACES SHOW DETAIL command. This command returns information on the “Minimum Recover Time,” which is the earliest point in time to which the table space can be rolled forward. The table space must be rolled forward to at least the Minimum Recover Time so that is synchronized with the information in the system catalog tables.

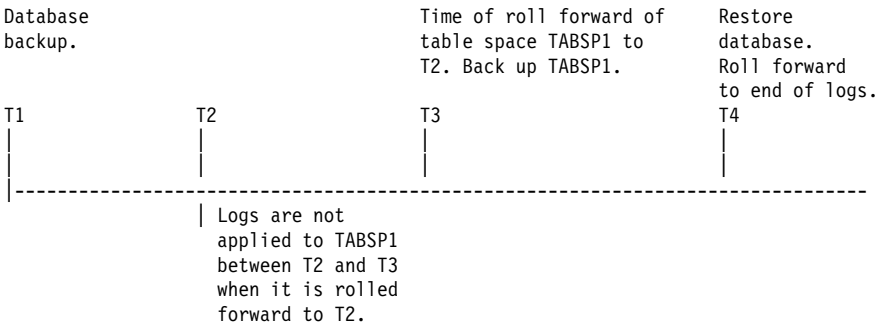
  The Minimum Recover Time is updated when DDL statements are executed against the table space, or against tables in the table space.

  Because the recovered table space must be consistent with the system catalog tables, you cannot perform a table space roll forward to recover a dropped table space or table, because the catalog table will indicate that the object was previously dropped. This means that you should not create dummy tables in those table spaces that you want to recover separately from the database.
- If you want to roll forward a table space to a point in time and a table in the table space participates in a referential integrity relationship with another table that is contained in another table space, you should roll forward both table spaces simultaneously to the same point in time. If you do not, both table spaces will be in the check pending state at the end of the point-in-time roll forward operation. If you roll forward both table spaces at the same time, the constraint will remain active at the end of the point-in-time roll forward operation.
- You should be careful that a point-in-time table space roll forward operation does not cause a transaction to be rolled back in some table spaces, and committed in others. This can happen when:
  - Point-in-time roll forward is performed on a subset of the table spaces that were updated by a transaction, and the point in time is before the time that the transaction committed.

– Any table contained in the table space being rolled forward to a point in time has an associated trigger, or is updated by a trigger that affects table spaces other than the one that is being rolled forward.

You should find a stop time that will prevent this from happening.

- After a table space point-in-time roll forward operation completes, the table space (or table spaces) is placed in the backup pending state. You must take a backup of the table space because the log records for the table space that are between the point in time that you rolled forward to and the current time are not applied. The following example shows why the table space backup is required, and how it is used. (To make the table space available, you can either back up the entire database, the table space that is in the backup pending state, or a set of table spaces that includes the table space that is in the backup pending state.)



In the preceding example, you back up the database at time T1. Then, at time T3, you roll forward table space TABSP1 to the point in time T2, then take a back up of the table space after T3. (Because the table space is in the backup pending state, you must take a backup of it. The timestamp of the table space backup is after T3, but the table space is at time T2. Log records are not applied to TABSP1 from between T2 and T3.) At time T4, you restore the database with the backup you took at T1 and roll forward to the end of the logs. The table space TABSP1 will be placed into the restore pending state when time T3 is reached.

The table space is put into the restore pending state at T3 because the database manager assumes that operations were performed on TABSP1 between T3 and T4 without the log changes between T2 and T3 having been applied to the table space. If the log changes between T2 and T3 were reapplied as part of the ROLLFORWARD on the database, this assumption would be violated. The required backup of a table space that must be taken after the table space is rolled forward to a point in time allows you to roll that table space forward past the time of the point-in-time roll forward (T3 in the example).

Assuming that you want to recover table space TABSP1 to T4, you would restore the table space from a backup that was taken after T3 (either the required backup, or a later one) then roll forward TABSP1 to the end of the logs.

In the preceding example, the most efficient way of restoring the database to time T4 would be to perform the required steps in the following order. Because you restore the table space before rolling forward the database, resource is not used to apply log records to the table space when the database is rolled forward, which

would happen if you rolled forward the database before you restored the table space.

1. Restore the database.
2. Restore the table space.
3. Roll forward the database.
4. Roll forward the table space.

If you cannot find back up image of TABSP1 that is after time T3, or you want to restore TABSP1 to T3 or before, you can:

- Roll forward the table space to the T3 point in time (in this situation, you do not need to restore the table space again).
- Restore the table space again from the backup of the database that you took at time T1, then roll forward the table space to a time that precedes time T3.
- Drop the table space.

Notes:

1. If, in a partitioned database environment, some database partitions are in the roll-forward pending state, and, on other database partitions, some table spaces are in the roll-forward pending state (but the database partition is not), you must first roll forward the database partitions, then roll forward the table spaces.
2. In a partitioned database environment you must roll forward all portions of the table space to the same point in time at the same time. This ensures that the table space is consistent at each database partition.

Planning to Use the ROLLFORWARD Command

Before using the ROLLFORWARD command you should consider the following items:

- You must have SYSADM, SYSCTRL, or SYSMAINT authority.
- The database may be local or remote.
- The database must be configured for roll-forward recovery (that is, either *logretain*, *userexit*, or both must be enabled).
- When a database is first configured for the roll-forward function, you must make a backup copy of it.
- A database must be restored successfully (using the RESTORE command) before it can be rolled forward; but a table space does not. A table space may be temporarily put into the roll-forward pending state, but not require a restore to fix it (for example, if a power interruption occurs).
- A database roll forward runs offline. The database is not available for use until the roll forward completes (either by reaching the end of the logs or through a STOP ROLLFORWARD). You can, however, perform an online roll forward of all table spaces except SYSCATSPACE. When you perform an online roll-forward operation on a table space, it is not available for use, but the other table spaces in the database are.
- When rolling forward, you should always issue ROLLFORWARD QUERY STATUS before issuing ROLLFORWARD STOP. This helps you ensure that the forward recovery reapplied changes to the correct point in time. After a ROLLFORWARD STOP, it is not possible to roll forward additional changes.

  - If you perform end-of-log forward recovery, the QUERY STATUS can indicate that a log file (or files) is missing if the point in time returned by QUERY STATUS is earlier than you expect.
  - If you perform point-in-time forward recovery, the QUERY STATUS will help you ensure that the roll forward is to the correct point.
- A table space requires roll-forward recovery if it is in a roll-forward pending state. It is in this state following a table space level restore or being taken off-line because of a media error.

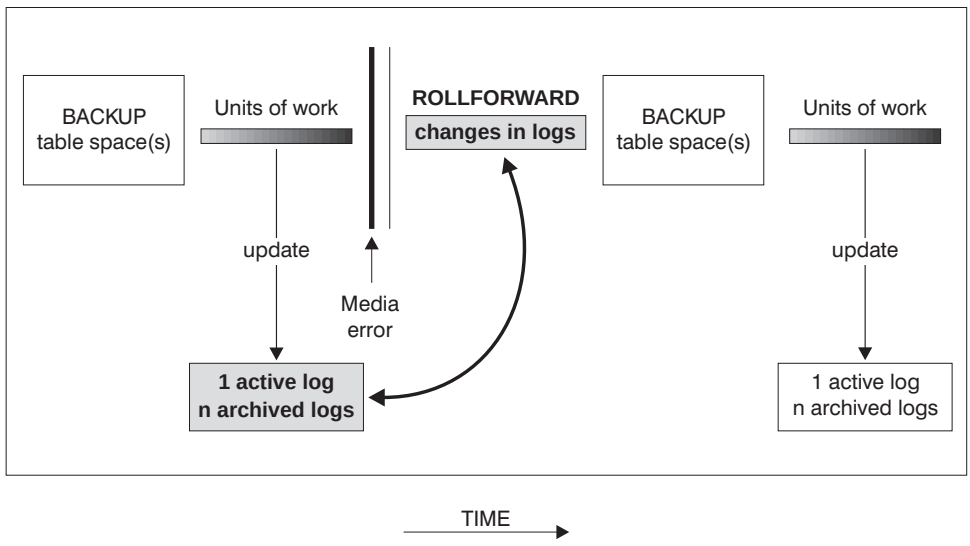


Figure 30. Table Space Roll-forward Recovery

- You can have multiple backups of a database or table space enabled for roll-forward recovery. You do not have to start with the latest backup copy of the database: you can start with any backup, as long as you have the logs associated with and following that backup.
- You should continue to make periodic backups of a database in order to reduce recovery time.
- If you need to stop a roll-forward operation to start it over again, you can use ROLLFORWARD CANCEL to cancel the operation.

  If you use ROLLFORWARD CANCEL against a database, this places the database into the restore pending state, whether or not a roll forward is in progress against the database.

  ROLLFORWARD CANCEL behavior for table spaces is as follows:
  - If you issue ROLLFORWARD CANCEL against a table space that is in the roll-forward pending state, the table space is put in the restore pending state. If you specify a list of table spaces that are in the roll-forward pending state, they are put in the restore pending state.

- If you issue ROLLFORWARD CANCEL against a table space that is in the rollforward-in-progress state and ROLLFORWARD is running to the end of the logs, the table space is put in the restore pending state. If multiple table spaces are being rolled forward to the end of the logs and you specify ROLLFORWARD CANCEL without a list, all table spaces that are in the rollforward-in-progress state are put in the restore pending state.
- If multiple table spaces are being rolled forward to the end of the logs and you specify ROLLFORWARD CANCEL with a list, the table spaces that are in the list are put in the restore pending state. The table spaces that are not in the list remain in the rollforward-in-progress state.
- If you issue ROLLFORWARD CANCEL and a table space is being rolled forward to a point in time, it is put in the restore pending state. If multiple table spaces are being rolled forward to a point in time and you do not specify a list, all the table spaces are put in the restore pending state. Even if you specify a list, the list is ignored and the table spaces are put in the restore pending state.

Invoking the ROLLFORWARD Command

There are a number of considerations before invoking the ROLLFORWARD command:

- You may specify a time when invoking the ROLLFORWARD command to limit the transactions that will be recovered from the database logs. If you are restoring from a backup that was created using the **online** option of the BACKUP command, the time on the ROLLFORWARD command must be later than the online backup end time.
- A log uses a timestamp associated with the completion of a unit of work. The timestamp in the logs uses the Coordinated Universal Time (CUT), which helps to avoid having the same timestamp associated with different logs (because of a change in time associated with daylight savings time, for example). The timestamp used on the backup is based on the local time that the BACKUP started. As a result, when you call the ROLLFORWARD command, you must specify the time in Coordinated Universal Time.

  Notes:

  1. The special register, CURRENT TIMEZONE, holds the difference between CUT and the local time at the application server database. Local time is the CUT plus the current timezone contents.
  2. If you are rolling forward a table space (or table spaces) to a point in time, you must roll forward at least to the minimum recovery time.
- If you stop the ROLLFORWARD task before it passes the point that the online backup ended, the database is left in a roll-forward pending state.

Using the Load Copy Location File

The DB2LOADREC environment variable is used to identify the file with the load copy location information. This file is used during roll-forward recovery to locate the load copy. It has information on:

- Media type

- Number of media devices to be used
- Location of the load copy generated during table load
- Filename of the load copy, if applicable

If the location file does not exist or no matching entry is found in the file, the information from the log record is used.

The information in the file may be overwritten before the roll-forward recovery takes place.

Notes:

1. In a partitioned database environment, the DB2LOADREC environment variable must be in the `db2profile` file.
2. In a partitioned database environment, the load copy file must exist at each database partition server, and the file name (including the path) must be the same.

The following information is provided in the location file. The first five parameters must have valid values and are used to identify the load copy. The entire structure is repeated for each load copy recorded. For example:

```
TIMestamp      19950725182542         * Timestamp generated at load time
SCHema         PAYROLL                * Schema of table loaded
TABlename      EMPLOYEES              * Table name
DATabasename   DBT                    * Database name
DB2instance    TORONTO                * DB2INSTANCE
BUFfernumber   NULL                   * Number of buffers to be used for recovery
SESsionnumber  NULL                   * Number of sessions to be used for recovery
TYPeofmedia    L                      * Type of media - L for local device
                                                       A for ADSM
                                                       O for other vendors
LOCationnumber 3                      * Number of locations
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
BUF            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
BUF            NULL
TYP            O
SHRlib         /@sys/lib/backup_vendor.a
```

Notes:

- The first 3 characters for each keyword are significant. All keywords are required in the specified order. No blank lines will be accepted.
- The timestamp is in the format *yyyymmddhhmmss*.
- All fields are mandatory except for BUF and SES which may be NULL. If SES is NULL, the value specified by configuration parameter NUMLOADRECSES will be used. If BUF is NULL, the default is SES+2.
- The type of media may be local device (L for tape, disk or diskettes), ADSM (A) or other vendor (O). If it is 'L', the number of locations followed by the location entries are required. If the type is 'A', no further input is required. If the type is 'O', the shared library name is required. For details about using ADSM and other vendor products as backup media, see "ADSTAR Distributed Storage Manager" on page 230.
- The SHRlib parameter points to a library that has function to store the LOAD COPY data.

Note: If you run LOAD COPY NO and do not take a backup copy of the database or affected table spaces after running LOAD, you cannot restore the database or table spaces to a point in time after the LOAD was performed. That is, you cannot use roll-forward recovery to rebuild the database or table spaces to a state after the LOAD. You can only restore the database or table spaces to a point in time that precedes the LOAD.

If you want to use a particular load copy, the LOAD timestamps are recorded in the recovery history file for the database. In a partitioned database environment, the recovery history file is local to each database partition.

For more information on LOAD, see "Using the LOAD Utility" on page 141.

Considerations for Managing Log Files

There are items to be considered when managing database logs:

- The numbering scheme for archived logs starts with S0000000.LOG, and goes through S9999999.LOG (10 000 000 logs). The database manager restarts using S0000000.LOG under these conditions:
  - When a database configuration file is changed to enable the roll-forward function.
  - When a database configuration file is changed to disable the roll-forward function.
  - When the logs wrap; that is, after log S9999999.LOG is used.

  When the roll-forward recovery method completes successfully, the last log is truncated, and logging begins with the next sequential log. The practical effect is that any log in the log path directory with a sequence number greater than the last log used for roll-forward recovery is re-used. You should keep a copy of the logs elsewhere if you want to be able to re-execute the ROLLFORWARD command using these old logs. (You may use a user exit program to copy the logs to another location.)

  You can have duplicate names for different logs because:

  - The database manager starts renaming logs with S0000000.LOG (as described above),
  - The database manager reuses log names after restoring a database (with or without roll-forward recovery).

  The database manager ensures that an incorrect log is not applied during roll-forward recovery, but it cannot detect the location of the required log. You must ensure that the correct logs are available for roll-forward recovery.
- If you moved log files to a location other than that specified by the *logpath* database configuration parameter, use the OVERFLOW LOG PATH parameter of the ROLLFORWARD command to specify the additional path to them.

  If you are rolling forward changes in a database or table space and the roll-forward operation cannot find the next log, the log name is returned in the SQLCA, indicating the next log file needed, and roll-forward recovery stops. At this time, if there are no more logs available, you can use the ROLLFORWARD command to stop processing.

  If you terminate the roll-forward recovery (by specifying the STOP option on the ROLLFORWARD command) and the log containing the completion of a transaction has not been applied to the database or table space, the incomplete transaction will be rolled back to ensure that the database or table space is left in a consistent state.
- Archived logs are placed in the SQLOGDIR subdirectory by default. To place them elsewhere, either enable the database for user exit, or use the OVERFLOW LOG PATH parameter of the ROLLFORWARD command to point to them when you roll forward.
- If you enable a user exit by changing the database configuration file, the archived logs can be redirected to a user-defined storage device such as a tape drive. Also, you can use a user exit program to manage the storage of archived logs. See Appendix J, “User Exit for Database Recovery” on page 753 for information about a user exit program.
- If you change the *newlogpath* parameter, any existing archived logs are unaffected. You must keep track of the location of the logs.
- If a database enabled for roll-forward recovery is restored either without being rolled forward or with being rolled forward to a specific time, an archived log may be associated with two or more different log sequences of a database, because log names are reused. (Figure 31 on page 229 provides an illustration of the logs that are created. If you now do a restore using “Backup 2” you must take extra care since there are two log sequences which could be used.) Before discarding an archived log, you must ensure that you do not need it.

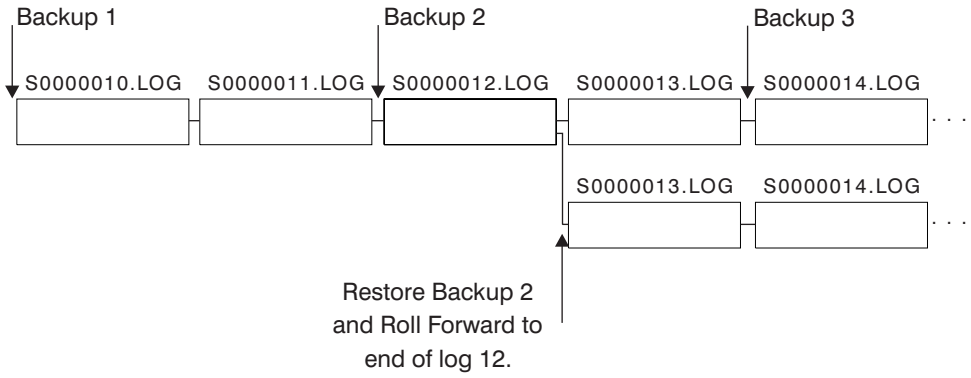


Figure 31. Reusing Log File Names

- If during a full database recovery you have rolled forward to a point in time and stopped in the middle of the logs, you have created a new log sequence. The two (2) log sequences cannot be combined. If you have an online backup that spans through the first log sequence, you must use the first log sequence to complete the roll forward recovery.
- If you have created a new log sequence after recovery, any table space backups taken in the old log sequence are invalidated. Restore rejects the table space backups in this case. There may be instances where restore fails to recognize that the backup is no longer valid (particularly for online backups) and the restore is successful. However, roll-forward for the table space will fail and the table space is left in a roll-forward pending state.

  In the diagram above, assume that a table space backup, Backup 3, is completed between `S0000013.LOG` and `S0000014.LOG` in the top log sequence. If we restored and rolled forward using database Backup 2, we would need to roll-forward through `S0000012.LOG`. After this we could continue to roll-forward through either the top log sequence or the newer bottom log sequence. If we rolled forward through the bottom sequence, we would not be able to use the table space Backup 3 to do a table space restore and roll-forward recovery.

  To be able to complete a table space roll-forward to end of logs using the table space Backup 3, we would have to restore using database Backup 2 and then roll-forward using the top log sequence. Once the table space Backup 3 has been restored, you can then request a roll-forward to end of logs.
- A log uses a timestamp associated with the completion of a unit of work. The timestamp in the logs uses the Coordinated Universal Time (CUT), which helps to avoid having the same timestamp associated with different logs (because of a change in time associated with daylight savings time, for example). The timestamp used on the backup is based on the local time. As a result, when you call the ROLLFORWARD command, you must specify the time in Coordinated Universal Time.

Note: The special register, CURRENT TIMEZONE, holds the difference between CUT and the local time at the application server database. Local time is the CUT plus the current timezone contents.

Losing Logs

- Dropping a database erases all logs in the current database log path directory. Before dropping a database, you may need to make copies of the logs.
- If you are rolling forward a database to a point-in-time, the last log used in the roll-forward recovery and all existing logs following that are reused. You lose the ability to recover past that particular point-in-time. Therefore, you should copy all the logs in the current database log path directory **before** beginning a point-in-time recovery.

  When the roll-forward processing completes, the log file with the last committed transaction is truncated, and logging begins with the next sequential log. If you do not have a copy of the log before it was truncated and those with higher sequence numbers, you cannot recover the database past the specified point-in-time. (Once normal database activity occurs following the roll-forward, new logs are created which can then be used in any subsequent recovery.)
- If you change the log path directory and then remove the subdirectory or erase any logs in that subdirectory called for in the log path, the database manager will look for the logs in the default log path, SQLOGDIR, when the database is opened. If the logs are not found, the database will enter a backup pending state, and you must back up the database before it is usable.

  This backup must be made even if the subdirectory contained empty logs.
- If you lose the log containing the point in time of the end of the online backup and you are rolling forward the corresponding restored image, the database will not be usable. To make the database usable, you must restore the database from a different backup and all associated logs.

You may encounter a situation similar to the following: You would like to do a point-in-point-in-time recovery on a full database but you are concerned that you might lose a log during the recovery process. (This scenario could occur if you have an extended number of archived logs between the time of the last backup database image and the point-in-time where you would like to have the database recovered.)

First, you should copy all of the applicable logs to a “safe” location. Then you can run the RESTORE command and use the roll-forward recovery method to the point-in-time you wish for the database. If any of the logs that you need is damaged or lost during this process, you have a backup copy of all of the logs elsewhere.

ADSTAR Distributed Storage Manager

When calling the BACKUP and RESTORE commands, you can specify that you want to use the ADSTAR Distributed Storage Manager (ADSM) product to manage the database or table space backup. The following topics provide additional information:

- Setting up an ADSTAR Distributed Storage Manager Client for UNIX-Based Platforms

- Setting up an ADSTAR Distributed Storage Manager Client for Other Platforms
- Considerations for Using ADSTAR Distributed Storage Manager

Setting up an ADSTAR Distributed Storage Manager Client for UNIX-Based Platforms

Before the database manager can use the ADSM option, the following set-up activities must be performed:

1. Set the environment variables used by ADSM:

   DSMI_DIR identifies the user-defined directory path where the API trusted agent file (`dsmapicta`) and the API message file (`dscameng.txt` and `dsgameng.txt`) are located.

   DSMI_CONFIG identifies the user-defined directory path to the `dsm.opt` file, which contains the ADSM user options.

   DSMI_LOG identifies the user-defined directory path where the error log (`dsierror.log`) will be created.

2. Create (or modify) the ADSM system configuration options file (`dsm.sys`).
3. Create (or modify) the `dsm.opt` ADSM user configuration options file. The environment variable `DSMI_CONFIG` points to this file.
4. Establish the ADSM password.

   For an ADSM client to be able to interface with an ADSM server, it must have a password for the server. The executable file `dsmapipw` is installed in the `INSTHOME/sqllib/adsm` directory of the instance owner. This executable allows you to establish and reset the ADSM password.

   To execute the `dsmapipw` command, you must be logged in as the “root” user. When this command is executed, you will be prompted for the following information:
   - *old password*, which is the current password for the ADSM node, as recognized by the ADSM server. The first time you execute this command, this password will be the one provided by the ADSM administrator at the time your node was registered on the ADSM server.
   - *new password*, which is the new password for the node that will be stored at the ADSM server. (Note that you will be prompted twice for the new password, to check for input errors.)

   Note: The user executing the BACKUP or RESTORE commands does not need to know this password. The only times you need to run this command are to establish a password for the initial connection and if the password has been reset on the ADSM server.

5. If the database manager is running, you should:
   - Stop the database manager using the `db2stop` command.
   - Start the database manager using the `db2start` command.

Setting up an ADSTAR Distributed Storage Manager Client for Other Platforms

Before the database manager can use the ADSM option, the following set-up activities must be performed:

1. Set the environment variables used by ADSM:

   DSMI_DIR identifies the user-defined directory path where the API trusted agent file (`dsmapicta`) and the API message file (`dscameng.txt`) are located.

   DSMI_CONFIG identifies the user-defined directory path to the `dsm.opt` file, which contains the ADSM user options.

   DSMI_LOG identifies the user-defined directory path where the error log (`dsierror.log`) will be created.

2. If applicable to your operating system, create (or modify) the ADSM system configuration options file (`dsm.sys`).
3. Create (or modify) the `dsm.opt` ADSM user configuration options file. The environment variable `DSMI_CONFIG` points to this file.
4. Establish the ADSM password.

   For an ADSM client to be able to interface with an ADSM server, it must have a password for the server. The executable file `dsmapipw` is installed in the `\sqllib\adsm` directory of the instance owner. This executable allows you to establish and reset the ADSM password.

   To execute the `dsmapipw` command, you must be logged in as the local administrator. When this command is executed, you will be prompted for the following information:
   - *old password*, which is the current password for the ADSM node, as recognized by the ADSM server. The first time you execute this command, this password will be the one provided by the ADSM administrator at the time your node was registered on the ADSM server.
   - *new password*, which is the new password for the node that will be stored at the ADSM server. (Note that you will be prompted twice for the new password, to check for input errors.)

   Note: The user executing the BACKUP or RESTORE commands does not need to know this password. The only times you need to run this command are to establish a password for the initial connection and if the password has been reset on the ADSM server.

5. If the database manager is running, you should:
   - Stop the database manager using the `db2stop` command.
   - Start the database manager using the `db2start` command.

Considerations for Using ADSTAR Distributed Storage Manager

To use specific features within ADSM, you may be required to give the fully-qualified path name of the object using the feature. (Remember that on Intel platforms the \ will be used instead of /.) The fully-qualified path name of:

- A full database backup object is:
  /<database>/NODEnnnn/FULL_BACKUP.timestamp.seq_no
- A table space backup object is:
  /<database>/NODEnnnn/TSP_BACKUP.timestamp.seq_no
- A load copy object is: /<database>/NODEnnnn/LOAD_COPY.timestamp.seq_no

where <database> is the database alias name, and NODEnnnn is the node number.

Note: The names shown in upper case must be entered as shown.

- In the case where you have multiple backups using the same database alias name, the timestamp and sequence number become the distinguishing part of the fully qualified name. You will need to query ADSM in order to determine which backup version to use.
- Individual backups are not known to the ADSM graphical user interface. Backup images are pooled into file spaces which ADSM manages. Individual backups can only be manipulated through the ADSM APIs, or through *db2adutl* which uses these APIs.
- The ADSM server will time-out a session if the ADSM client does not respond for the period of time specified by the COMMTIMEOUT parameter in the server's configuration file. Three factors may contribute to the occurrence of this timeout problem:
  - The COMMTIMEOUT parameter is set too low at the ADSM server. For example, during a restore, if large DMS table spaces are being created, a timeout may occur.

    The recommended value for this parameter is 6000 seconds.
  - The database manager backup (or restore) buffer is too large.
  - The database activity is too high during an online backup.
- The database manager uses the full backup option of ADSM; ADSM incremental backups are not supported
- Use multiple sessions to increase throughput.

Managing Backups and Log Archives on ADSM

The **db2adutl** utility allows you to query, extract, and delete backups, logs, and load copy images saved using ADSM. The utility is installed in the INSTHOME/sqllib/misc directory on UNIX platforms and in the \sqllib\misc directory on Intel platforms.

All of the options available through the db2adutl utility are shown:

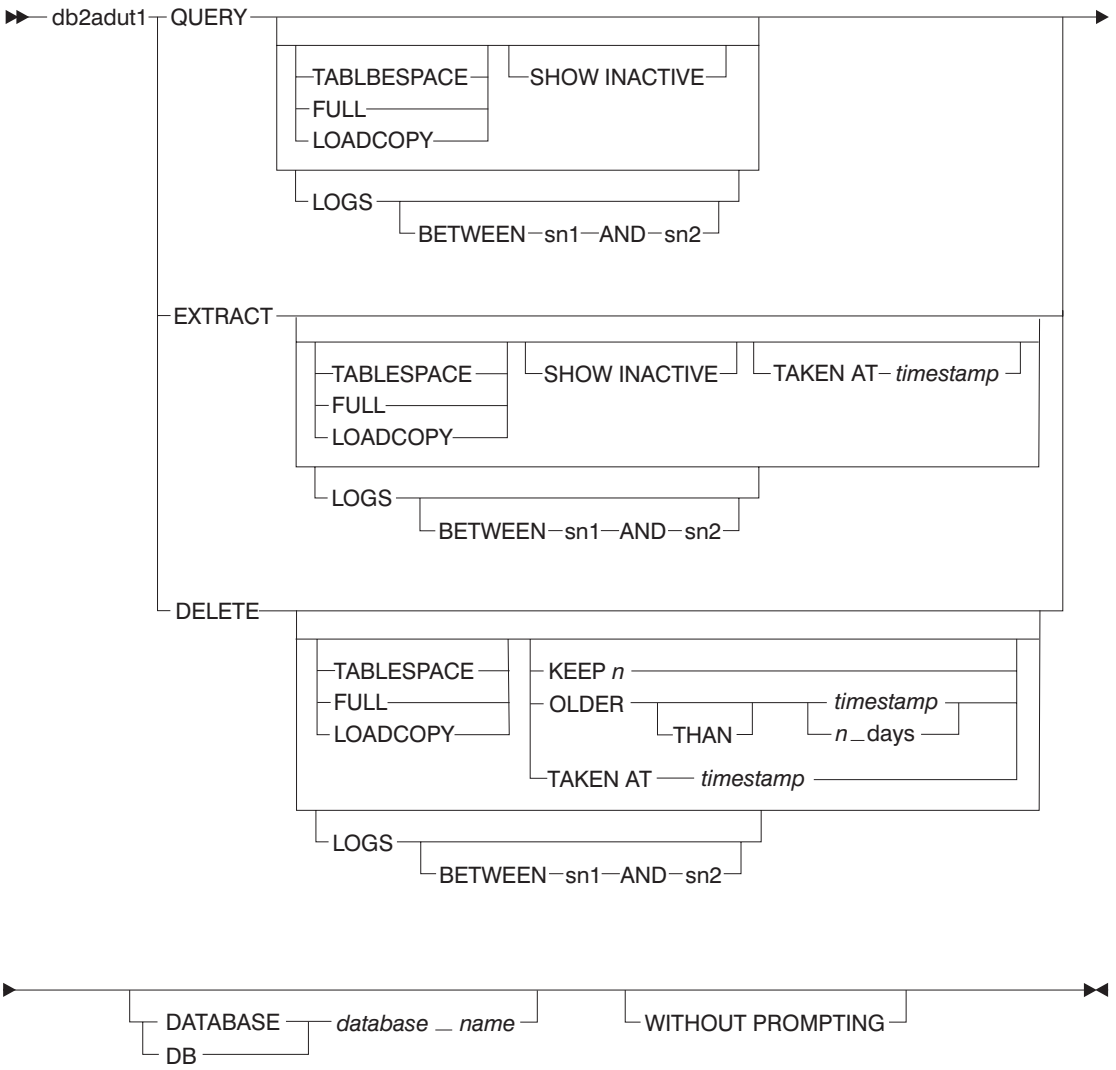


Figure 32. Syntax for db2adutl

You can choose which database you wish to work with when you use each command through the use of the DATABASE parameter. For the EXTRACT and DELETE commands, you can request not to see the prompts to confirm your choices through use of the WITHOUT PROMPTING parameter.

The QUERY command of this utility allows you to list backups. logs, and load copy images. The backups can be full database, table spaces, or both. When using this command, the default is to list both types of backups, any load copy images, and any logs. You can select a range of logs to be listed instead of seeing all of the logs. You can also request to see the inactive backups.

The EXTRACT command of this utility allows you to copy from ADSM to your current directory backups, logs, or both at the ADSM server. The backups can be full database, table spaces, or both. When using this command, the default without qualifiers is to list the active backups and each log. You can then select which backups and/or logs to extract. You can also select a range of logs to be listed instead of seeing all of the logs. You can also request to see the inactive backups. A specific backup for extraction can be selected by using the TAKEN AT <timestamp> parameter.

The DELETE command of this utility allows you to delete logs or deactivate backups from ADSM. When using this command, the default without qualifiers is to list the active backups and each log. You can then select which backups and/or logs to delete/deactivate. You can qualify the command with KEEP n to keep the most recent n backups. You can also qualify the command with OLDER [THAN] <timestamp> or n DAYS. This will delete backups older than the given date (timestamp) or older than the days specified. You can also select a range of logs to be listed instead of seeing all of the logs. A specific backup for deletion can be selected by using the TAKEN AT <timestamp> parameter.

For DB2, we recommend that the ADSM default policy be used. With the changes to the backup naming conventions, each backup is now unique. In order to delete old backups, the policy must be set up so that no active copies are kept.

For examples of using this utility, see “Examples of Using db2adutl.”

Examples of Using db2adutl:

```
db2 backup database rawsampl use adsm

Backup successful. The timestamp for this backup is : 19970929130942

db2adutl query

Query for database RAWSAMPL

Retrieving full database backup information.
   full database backup image: 1, Time: 19970929130942, Oldest log: S0000053.LOG, Sessions used: 1
   full database backup image: 2, Time: 19970929142241, Oldest log: S0000054.LOG, Sessions used: 1

Retrieving tablespace backup information.
   tablespace backup image: 1, Time: 19970929094003, Oldest log: S0000051.LOG, Sessions used: 1
   tablespace backup image: 2, Time: 19970929093043, Oldest log: S0000050.LOG, Sessions used: 1
   tablespace backup image: 3, Time: 19970929105905, Oldest log: S0000052.LOG, Sessions used: 1

Retrieving log archive information.
   Log file: S0000050.LOG
   Log file: S0000051.LOG
   Log file: S0000052.LOG
   Log file: S0000053.LOG
   Log file: S0000054.LOG
   Log file: S0000055.LOG
```

```
db2adutl delete full taken at 19950929130942 db rawsampl

Query for database RAWSAMPL

Retrieving full database backup information.  Please wait.

   full database backup image: RAWSAMPL.0.db26000.0.19970929130942.001

   Do you want to deactivate this backup image (Y/N)? y

     Are you sure (Y/N)? y
db2adutl query

Query for database RAWSAMPL

Retrieving full database backup information.
   full database backup image: 2, Time: 19950929142241, Oldest log: S0000054.LOG, Sessions used: 1

Retrieving tablespace backup information.
   tablespace backup image: 1, Time: 19950929094003, Oldest log: S0000051.LOG, Sessions used: 1
   tablespace backup image: 2, Time: 19950929093043, Oldest log: S0000050.LOG, Sessions used: 1
   tablespace backup image: 3, Time: 19950929105905, Oldest log: S0000052.LOG, Sessions used: 1

Retrieving log archive information.
   Log file: S0000050.LOG
   Log file: S0000051.LOG
   Log file: S0000052.LOG
   Log file: S0000053.LOG
   Log file: S0000054.LOG
   Log file: S0000055.LOG
```

Part 2. Distributed Transaction Processing

Chapter 7. Distributed Databases

In the DB2 database manager, a transaction is commonly referred to as a *unit of work*. A unit of work is a recoverable sequence of operations within an application process, and is the basic building block used by the database manager to ensure that a database is in a consistent state. Any reading or writing to the database is done within a unit of work. A *point of consistency* (or commit point) is a time when all recoverable data that an application accesses is consistent with related data.

For example, a bank transaction might involve the transfer of funds from a savings account to a checking account. After the application subtracts the amount from the savings account, the two accounts are inconsistent; they are not consistent again until the amount is added to the checking account. When both steps are completed, the point of consistency is reached, and the changes can be committed and are made available to other applications.

A unit of work starts when the first SQL statement is issued against the database. The application must end the unit of work by issuing either a COMMIT or a ROLLBACK statement. The COMMIT statement makes permanent all changes made within a unit of work, whereas the ROLLBACK statement removes these changes from the database. If the application ends normally without either of these statements, the unit of work is automatically committed. If it ends abnormally while in the middle of a unit of work, the unit of work is automatically rolled back. Once issued, a COMMIT or ROLLBACK cannot be stopped. With some multi-threaded applications, if the application ends normally without either of these statements, the unit of work is automatically rolled back. Similarly on some operating systems, if the application ends normally without either of these statements, the unit of work is automatically rolled back. The recommendation when writing your applications is to always explicitly COMMIT or ROLLBACK your completed unit of work.

In the above banking example, only if both requests are processed successfully, should the application direct the database manager to commit the changes. If either request is not processed successfully, the updates should be rolled back, leaving both tables as they were before the transaction began. This ensures that requests are neither lost nor duplicated.

The following topics provide additional information:

For information on creating applications using distributed databases, see the *Embedded SQL Programming Guide* and the *CLI Guide and Reference* manuals.

Using a Single Database in a Transaction

The simplest form of database usage is to read and write to only one database within a single transaction (unit of work). This type of database access is called *remote unit of work*.

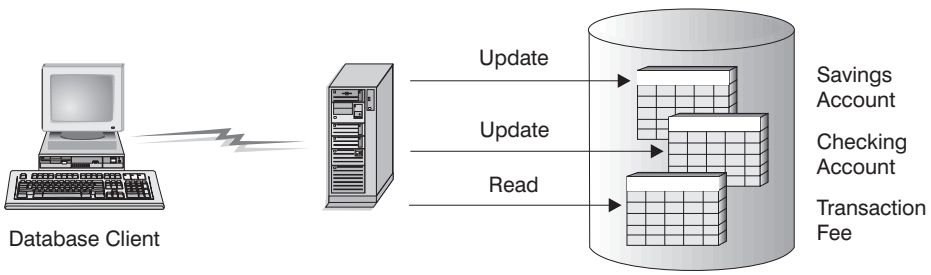


Figure 33. Using a Single Database in a Transaction

Figure 33 shows an example of a database client running a funds transfer application that accesses a database containing checking and savings account tables, as well as a banking fee schedule. The application performing the transfer includes the following steps:

1. Accept the amount to transfer from the user interface
2. Subtract the amount from the savings account and determine the new balance
3. Read the fee schedule to determine the transaction fee for a savings account with the given balance
4. Subtract the transaction fee from the savings account
5. Add the amount of the transfer to the checking account
6. Commit the transaction (unit of work).

To set up this funds transfer application, you must:

1. Create the tables for the savings account, checking account and banking fee schedule in the same database (Chapter 3, “Implementing Your Design” on page 55)
2. (If physically remote...) Set up the database server to use the appropriate communications protocol, as described in the *Quick Beginnings* manuals
3. (If physically remote...) Catalog the node and database to identify the database on the above database server, as described in the *Quick Beginnings* manuals
4. Pre-compile your application program to specify a type 1 connection, that is, specify CONNECT(1) on the PREP command, as described in the *Embedded SQL Programming Guide* manual.

Using Multiple Databases in a Single Transaction

When using multiple databases in a single transaction, the requirements for setting up and administering your environment are different, depending on the number of databases that are being updated in the transaction. For more information, see:

- "Updating a Single Database"
- "Updating Multiple Databases" on page 242.

Updating a Single Database

If your data is distributed across multiple databases, you may wish to update one database while reading from one or more other databases. This type of access can be performed within a single unit of work (transaction). This type of database access is called *distributed unit of work*. See "Updating Multiple Databases" on page 242 for another example of distributed unit of work.

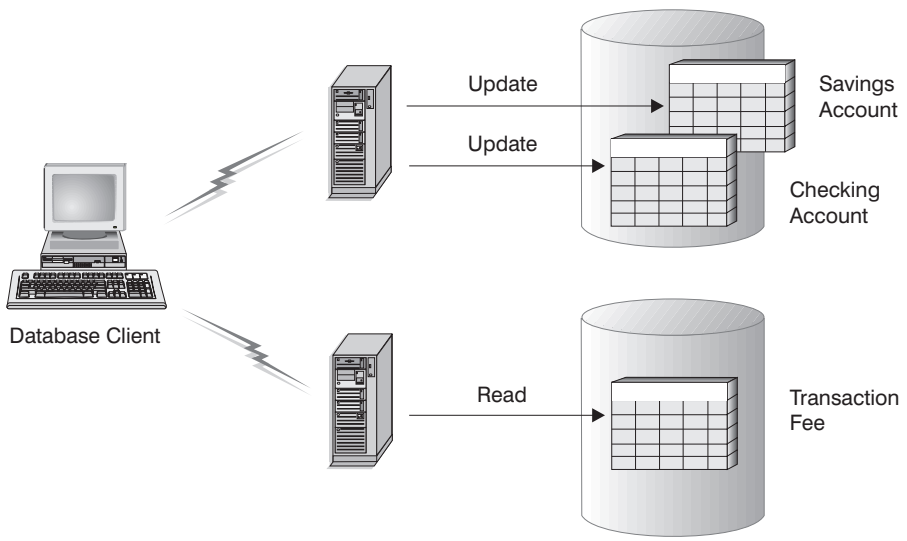


Figure 34. Using Multiple Databases in a Single Transaction

Figure 34 shows an example of a database client running a funds transfer application that accesses two database servers: one containing the checking and savings accounts and another containing the banking fee schedule. This example is similar to the example provided in "Using a Single Database in a Transaction" on page 240, except for the number of databases and the location of the tables. As discussed previously, the application performing the transfer includes the following steps:

1. Accept the amount to transfer from the user interface
2. Subtract the amount from the savings account and determine the new balance
3. Read the fee schedule to determine the transaction fee for a savings account with the given balance

4. Subtract the transaction fee from the savings account
5. Add the amount of the transfer to the checking account
6. Commit the transaction (unit of work).

To set up the above environment, you must:

1. Create the necessary tables in the appropriate databases (Chapter 3, “Implementing Your Design” on page 55)
2. (If physically remote...) Set up the database servers to use the appropriate communications protocols, as described in the *Quick Beginnings* manuals
3. (If physically remote...) Catalog the nodes and databases to identify the databases on the above database servers, as described in the *Quick Beginnings* manuals
4. Pre-compile your application program, as described in the *Embedded SQL Programming Guide* to specify:
    a. A type 2 connection, that is, specify CONNECT(2) on the PREP command
    b. One-phase commit, that is SYNCPOINT(ONEPHASE) on the PREP command.

Performance Tip: You should note that unlike the scenario described in “Updating Multiple Databases” updating a single database while reading multiple databases only requires a one-phase commit (SYNCPOINT(ONEPHASE) on PREP command). Using a one-phase commit process requires less overhead than a two-phase commit process. Therefore, performance is better when using SYNCPOINT(ONEPHASE) rather than SYNCPOINT(TWOPHASE) for applications that only update a single database within a unit of work.

DRDA Application Server Additional Information:

- If the databases containing the tables used in the above example are located on DB2 for MVS/ESA, OS/390, OS/400, VM or VSE host systems, then the DB2 Connect product is needed. See the *DB2 Connect User's Guide* for additional information on how to set up and use DB2 Connect.
- Chapter 8, “Using DB2 with an XA-Compliant Transaction Manager” on page 255 provides information about using your database in an environment with a transaction processing monitor, such as CICS.

Updating Multiple Databases

If your data is distributed across multiple databases, you may also wish to read and update several databases in a single transaction. This type of database access is called *distributed unit of work*. This type of environment is more complex than that described in “Updating a Single Database” on page 241. As a result, additional topics will be introduced below.

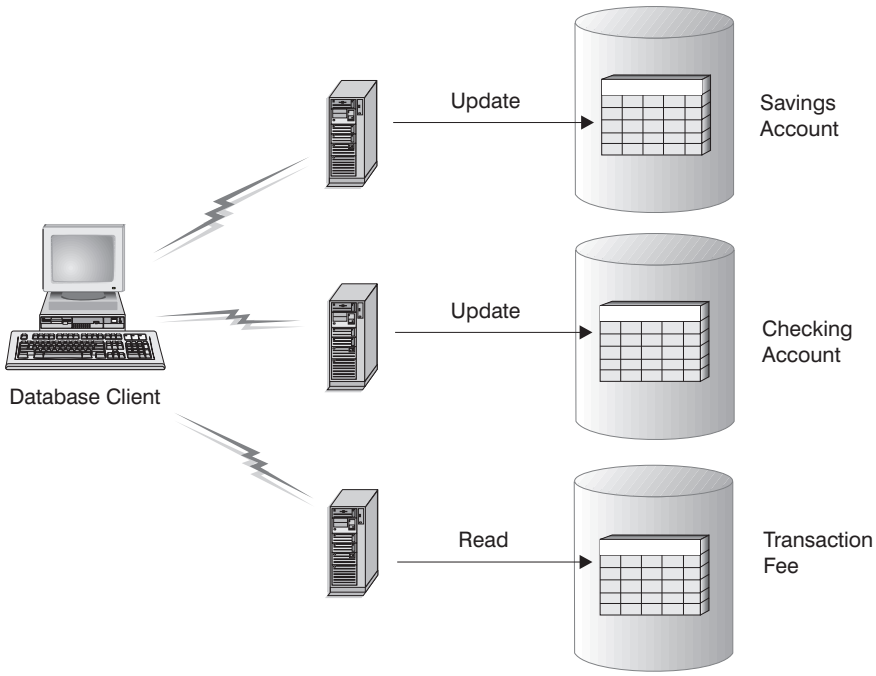


Figure 35. Updating Multiple Databases

Figure 35 shows an example similar to Figure 34 on page 241, except the checking and savings accounts are located in different databases. The application performing the transfer includes the same steps as described in "Updating a Single Database" on page 241.

1. Accept the amount to transfer from the user interface
2. Subtract the amount from the savings account and determine the new balance
3. Read the fee schedule to determine the transaction fee for a savings account with the given balance
4. Subtract the transaction fee from the savings account
5. Add the amount of the transfer to the checking account
6. Commit the transaction (unit of work).

To set up the above environment, you must:

1. Create the necessary tables in the appropriate databases (Chapter 3, "Implementing Your Design" on page 55)
2. (If physically remote...) Set up the database servers to use the appropriate communications protocols, as described in the *Quick Beginnings* manuals

3. (If physically remote...) Catalog the nodes and databases to identify the databases on the above database servers, as described in the *Quick Beginnings* manuals
4. Pre-compile your application program, as described in the *Embedded SQL Programming Guide* to specify:
   a. A type 2 connection, that is, specify CONNECT(2) on the PREP command
   b. Two-phase commit, that is SYNCPOINT(TWOPHASE) on the PREP command.
5. Configure the DB2 transaction manager (TM), as described in "Using the DB2 Transaction Manager." This section also provides information about how the two-phase commit process works.

Using the DB2 Transaction Manager

The database manager provides transaction manager functions that can be used to coordinate updating several databases within a single unit of work. The database client automatically coordinates the unit of work and uses a *transaction manager database* to register each transaction (unit of work) and to track the completion status of that transaction.

The database that will be used as the transaction manager database is determined at the database client by the database manager configuration parameter *tm_database* (see "Transaction Manager Database Name (tm_database)" on page 533). Consider the following factors when setting this configuration parameter:

- The transaction manager database can be:
  - Another DB2 database.
  - A DB2 for OS/390 Version 5 or later database.
- The transaction manager database can be any database.
- Catalog databases and nodes to allow the following:
  - All database manager instances participating in a distributed transaction must be able to connect to the transaction manager database that was specified by the client's *tm_database* configuration parameter. An instance participates in a distributed transaction if the transaction connects to one or more databases contained in that instance. If, for example, the *tm_database* parameter is set to `DB2TRMGR` at the database client, you should be able to issue the following command from each participating instance:

    ```
    CONNECT TO DB2TRMGR
    ```

    The result of this command should connect you to the same database, on the same node from every participating instance, as well as the database client.
  - The database manager instance containing the transaction manager database must be able to connect to all other databases participating in the distributed transaction. If, for example, the client connects to the SAVINGS_DB, CHECKING_DB and FEE_DB, the instance containing the transaction manager database must also be able to connect to those databases using the same names or aliases that the database client uses.

Note: The transaction manager database must not be cataloged using the alias option to specify an alternate name.

- If the keyword `1ST_CONN` is defined for the *tm_database* parameter, the first database to which the application connects in the transaction will be used as the transaction manager database. In this case, all databases used in any transaction initiated from the database client must be able to connect to one another using the same database aliases as are used at the database client. This effectively means that each database within a network must have a unique alias across the network.

  Care must be taken when using `1ST_CONN` and you should only use this configuration if it is easy to maintain, for example, in the following situations:

  – The database client initiating the transaction is in the same instance that contains the participating databases, including the transaction manager database
  – You are using DCE directory services to catalog and manage access to your databases.

  Note that if your application attempts to disconnect from the database being used as the transaction manager database, you will receive a warning message and the connection will be held until the unit of work is committed.

The above rules regarding cataloging of aliases affect your ability to recover from problems (see “Recovering from Problems During Two-Phase Commit” on page 249).

Configuration Considerations

You should consider the values of the following configuration parameters when you are setting up your environment:

- The following are database manager configuration parameters:
  – “Transaction Manager Database Name (tm_database)” on page 533

    The *tm_database* database manager configuration parameter identifies the name of the Transaction Manager (TM) database for each DB2 instance.
  – “Sync Point Manager Log File Size (spm_log_file_sz)” on page 535

    The *spm_name* database manager configuration parameter identifies the name of the SNA Sync Point Manager (SPM) instance to the database manager. It is defined in the system database directory and, if remote, in the node directory.
  – “Transaction Resync Interval (resync_interval)” on page 534

    The *resync_interval* database manager configuration parameter identifies the time interval in seconds for which a Transaction Manager (TM), Resource Manager (RM), or Sync Point Manager (SPM) should retry the recovery of any outstanding indoubt transactions.
  – “Sync Point Manager Name (spm_name)” on page 534

    The *spm_log_file_sz* database manager configuration parameter identifies the SPM log file size in 4K pages.
  – “Sync Point Manager Resync Agent Limit (spm_max_resync)” on page 536

The *spm_max_resync* database manager configuration parameter identifies the number of agents that can simultaneously perform resync operations.

- The following are database configuration parameters:
  - "Maximum Number of Active Applications (maxappls)" on page 508

    The *maxappls* database configuration parameter specifies the maximum number of active applications allowed.

    The value of this parameter must be large enough to accommodate the application (user) connections to the database, as well as the number of indoubt units of work and the number of user transactions that have entered, but not yet completed, a two-phase syncpoint process.

    As a result, if you have an environment like the one just described, you may need to increase the value of the *maxappls* parameter. Increasing the value helps ensure that all processes can be accommodated.
  - "Auto Restart Enable (autorestart)" on page 529

    This database configuration parameter specifies whether the RESTART DATABASE routine will be invoked when needed. The default is yes (that is, enabled).

    A database containing indoubt transactions will require the RESTART DATABASE command/routine to be invoked in order to start up. If the *autorestart* option is not enabled, when the last connection to the database is dropped, the next connection will fail and require an explicit RESTART DATABASE again. This condition will exist until the indoubt transaction has been removed either by the transaction manager's resync operation, or through a heuristic operation performed by the administrator. When the RESTART DATABASE is issued, a message will be displayed if there are any indoubt transactions in the database. The administrator can then use the LIST INDOUBT TRANSACTION command and other command line processor commands to find out information about those indoubt transactions.

Understanding the Two-Phase Commit Process

The following example illustrates the steps of the example transaction (described in "Updating Multiple Databases" on page 242) and the participants in the transaction. If an error occurs during the two-phase commit process, understanding how a transaction is managed will help you to resolve the problem.

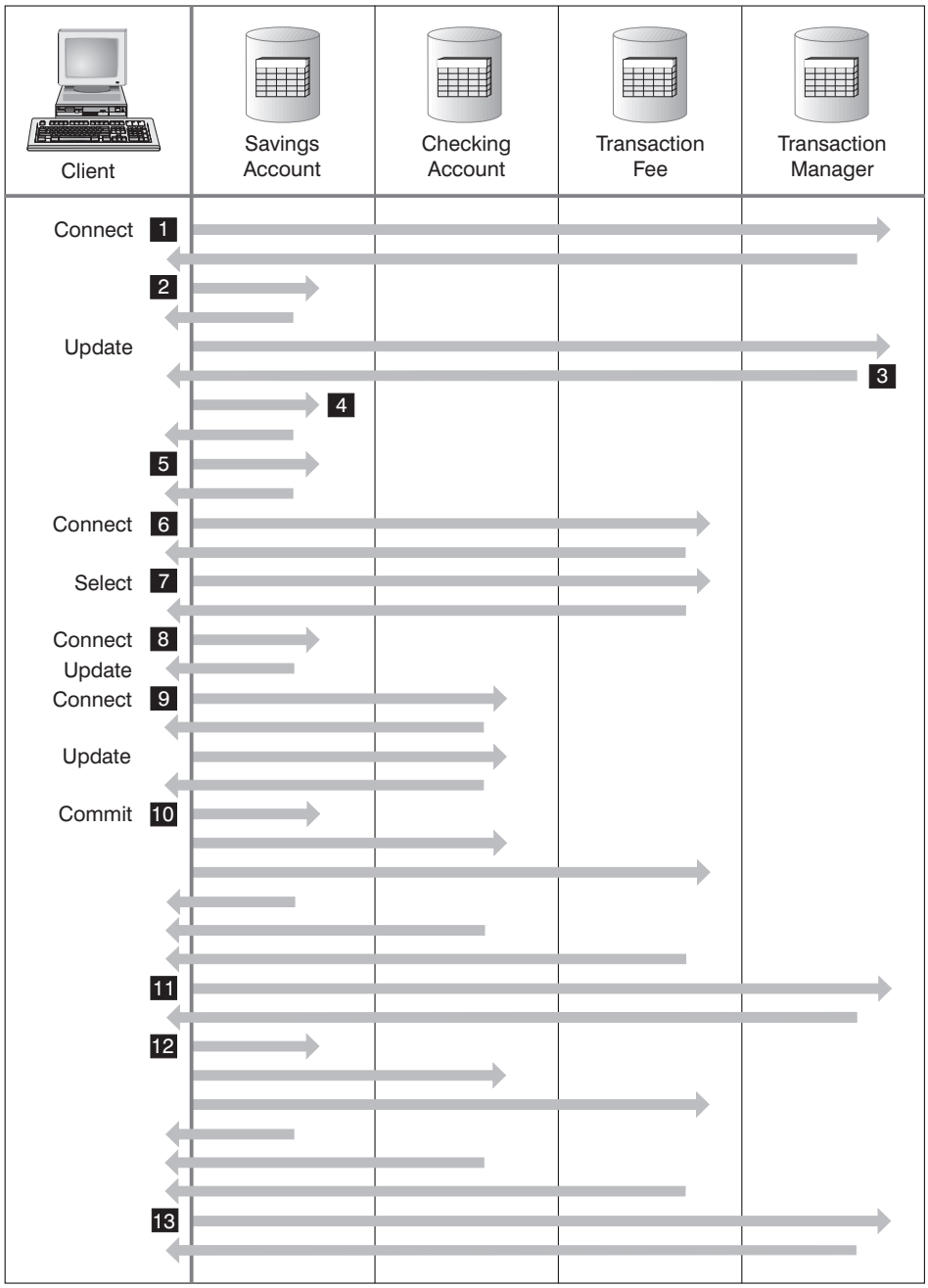


Figure 36. Updating Multiple Databases

1. When the database client wants to connect to SAVINGS_DB, it first connects to the Transaction Manager (TM) database. The TM database returns an acknowledgement to the database client.
2. The connection to the SAVINGS_DB takes place and is acknowledged.
3. The database client begins the update to the SAVINGS_ACCOUNT table. This begins the unit of work. The TM database responds to the database client providing a transaction ID for the unit of work. Note that the registration of a unit of work occurs when the first SQL statement in the unit of work is run, not necessarily during connect time.
4. After receiving the transaction ID, the database client registers the unit of work with the database containing the SAVINGS_ACCOUNT table. A response is sent back to the client to indicate that the unit of work has been registered successfully.
5. SQL statements issued against the SAVINGS_DB are handled in the normal manner. The response to each statement is returned in the SQLCA when working with SQL statements embedded in a program. (The SQLCA is described in the *Embedded SQL Programming Guide* and the *SQL Reference*.)
6. The transaction ID is registered at the FEE_DB database containing the TRANSACTION_FEE table, during the first access to that database within the unit of work.
7. Any SQL statements against the FEE_DB database are handled in the normal fashion.
8. Additional SQL statements can be executed against the SAVINGS_DB by setting the connection as appropriate. Since the unit of work has already been registered with the SAVINGS_DB 4, the database client does not need to perform the registration step again.
9. Connecting to and using the CHECKING_DB follows the same rules as described by 6 and 7.
10. When the database client requests that the unit of work be committed, a **prepare** message is sent to all databases participating in the unit of work. Each database writes a “PREPARED” record to their log files and replies to the database client.
11. After the database client receives a positive response from all of the databases, it sends a message to the transaction manager database to inform it that the unit of work is now ready to be committed (PREPARED). The transaction manager database writes a “PREPARED” record to its log file and sends a reply to inform the client that the second phase of the commit process can be started.
12. During the second phase of the commit process, the database client sends a message to all participating databases to tell them to commit. Each database writes a “COMMITTED” record to its log file and releases the locks that were held for this unit of work. When the database has completed committing the changes, it sends a reply to client.
13. After the database client receives a positive response from all participating databases, it sends a request to the transaction manager database to inform it that the unit of work has been completed. The transaction manager database then:

- Writes a “COMMITTED” record to its log file, to indicate that the unit of work is complete
- Replies to the client to indicate it has finished.

Recovering from Problems During Two-Phase Commit

Recovering from error situations is a normal task associated with application programming, system administration, database administration and system operation. Distributing databases over several remote servers increases the potential for error situations resulting from network or communication failures. To ensure data integrity, the database manager provides the two-phase commit process which is illustrated in “Updating Multiple Databases” on page 242 as points **10**, **11** and **12**. The following explain how the database manager handles errors during this two-phase commit process:

- First Phase Error

  If a database responds that it failed to prepare to commit the unit of work, the database client will roll back the unit of work during the second phase of the commit process. A prepare message will **not** be sent to the transaction manager database in this case.

  During the second commit-phase, the client sends a rollback message to all participating databases that successfully prepared to commit in the first phase. Each database then writes “ABORT” record to their log file and releases the locks held for this unit of work.
- Second Phase Error

  Error handling at this stage is dependent on whether the second phase will commit or roll back the transaction. The second phase will only roll back the transaction if the first phase encountered an error.

  If one of the participating databases fails to commit the unit of work (possibly due to a communications failure), the transaction manager database will retry the commit on the failed database. The database manager configuration parameter *resync_interval* (“Transaction Resync Interval (resync_interval)” on page 534) is used to determine how long the transaction manager database will wait between attempts to commit the unit of work.

  If the transaction manager database fails, it will resynchronize the unit of work when the database is restarted. The resynchronization process will attempt to complete all *indoubt transactions*, that is, those transactions that have finished the first phase and have not completed the second phase of the commit. The database manager, where the transaction manager database resides, will perform the resynchronization by:
  1. Connecting to the databases that replied that they were “PREPARED” to commit during the first phase of the commit process.
  2. Attempting to commit the indoubt transaction at that database. (If the indoubt transaction cannot be found, the database manager assumes that the database successfully committed the transaction during the second phase of the commit process.)

3. Committing the indoubt transaction in the transaction manager database, after all indoubt transactions have been committed in the participating databases.

If one of the participating databases fails and is restarted, the database manager for this database will check the log of the transaction manager database to determine whether the transaction should be rolled back. If the transaction is not found in the log, the database manager assumes the transaction was rolled back and will roll back the indoubt transaction for this database. Otherwise, the database will wait for a commit request from the transaction manager database.

Manual Recovery of Indoubt Transactions

If, for some reason, you cannot wait for the transaction manager to automatically resolve the indoubt transaction, there are some actions you can take to manually resolve the states of indoubt transactions. This manual process is sometimes referred to as “making a heuristic decision.”

The LIST INDOUBT TRANSACTIONS command (and a related set of APIs) allows you to query, commit, and roll back indoubt transactions. In addition, it also allows you to “forget” transactions that have been heuristically committed or rolled back by removing the log records and releasing the log space. For information about the command and APIs, see the *Command Reference* and the *API Reference* manuals.

You should use this command (or APIs) with **extreme caution** and as a last resort. The best solution is to wait for the transaction manager to drive the resynchronization process. You could experience data integrity problems, if you manually commit or roll back a transaction in one of the participating databases, and the opposite action is taken for another of the databases. Recovering from data integrity problems requires you to understand the application logic, the data changed or rolled back, and then to perform a point-in-time recovery of the database, or manually undo/redo the database changes.

If you cannot wait for the transaction manager to initiate the resynchronization process and you must release the resources tied up by an indoubt transaction, then heuristic operations are necessary. This situation could occur if the transaction manager will not be available for an extended period of time to do the resync, and the indoubt transaction is tying up resources that are urgently needed. An indoubt transaction ties up the resources that were associated with this transaction before the transaction manager or participating database became unavailable. These resources include things such as the locks on tables and indexes, log space, and storage taken up by the transaction. Each indoubt transaction also decreases (by one) the maximum number of concurrent transactions that can be handled by the database.

There are no foolproof ways to perform heuristic operations. You can use the following steps as a guideline:

1. Connect to the database for which you require all transactions to be complete.
2. Use the LIST INDOUBT TRANSACTIONS command to display the indoubt transactions. The **xid** represents the global transaction ID and is identical in other databases participating in this transaction, including the transaction manager database.

3. For each indoubt transaction, use your knowledge about the application and the *tm_database* configuration parameter to determine the transaction manager database as well as the other participating databases.
4. Connect to the transaction manager database.
   - If you were able to connect to this database, use the LIST INDOUBT TRANSACTIONS command to display the indoubt transactions recorded in the transaction manager database.
     – If there is an indoubt transaction with the same **xid** as recorded in step 2 and with type TM, you can connect to each database participating in the transaction, and heuristically commit the transaction using the LIST INDOUBT TRANSACTIONS command.
     – If there is **not** an indoubt transaction with the same **xid** as recorded in step 2 and with type TM, you can connect to the each database participating in the transaction, and heuristically roll back the transaction using the LIST INDOUBT TRANSACTIONS command.
   - If you **cannot** connect to the transaction manager database, you will have to use the status of the transaction in the other participating databases to determine what action you should take.
     – If at least one of the other databases has committed the transaction, then you should heuristically commit the transaction in all the participating databases (using LIST INDOUBT TRANSACTIONS command).
     – If at least one of the other databases has rolled back the transaction, then you should heuristically roll back the transaction in all the participating databases (using LIST INDOUBT TRANSACTIONS command).
     – If the transaction is in “PREPARED” (indoubt) state in all of the participating databases, then you should heuristically roll back the transaction in all the participating databases (using LIST INDOUBT TRANSACTIONS command).
     – If you are unable to connect to one or more of the other participating databases, then you should heuristically roll back the transaction in all the participating databases (using LIST INDOUBT TRANSACTIONS command).

Note: The LIST INDOUBT TRANSACTIONS command includes “type” information to show you the role of the database in each indoubt transaction:

TM indicates the indoubt transaction is using the database as a transaction manager database.

RM indicates the indoubt transaction is using the database as a resource manager, meaning that it is one of the databases participating in the transaction, but is not the transaction manager database.

Resynchronizing Indoubt Transactions if AUTORESTART=OFF

For configuration considerations in the DB2 Universal Database two-phase commit environment, refer to “Configuration Considerations” on page 245.

In particular, if the *autorestart* database configuration parameter is OFF and there are indoubt transactions in either the TM or RM databases, the RESTART DATABASE command is required in order to start up the resynchronization process. If issuing the

RESTART DATABASE command from the command line processor, use different sessions. If you restart a different database from the same session, the connection established by the previous restart database command will be dropped. The database will need to be restarted again if the last connection to it is dropped. Issue DB2 TERMINATE to drop the connection after there are no indoubt transactions listed by the DB2 LIST INDOUBT TRANSACTIONS command.

Recovery of Indoubt DRDA Transactions

There are some differences in how indoubt DRDA transactions are recovered as a result of the type of communications environment used.

Recovery Using SNA Communications

Recovery of indoubt DRDA transactions at "downstream" DRDA 2 Application Servers (AS) is normally performed automatically by the Transaction Manager (TM) and the LU 6.2 Sync Point Manager (SPM) if in an SNA communications environment. An indoubt transaction at a DRDA 2 AS does not hold any resources at the local DB2 location but does hold resources at the AS as long the transaction is indoubt at the AS. If the AS determines that a heuristic decision must be made, then the AS administrator may contact the local DB2 database administrator (for example via telephone) in order to determine whether to commit or roll back the transaction at the AS. If this occurs, the LIST DRDA INDOUBT TRANSACTIONS command can be used to determine the state of the transaction at the local DB2. The following steps can be used as a guideline for most situations involving a SNA communications environment.

1. Connect to the SPM as shown below.

```
db2 => connect to db2spm

 Database Connection Information

 Database product       = SPM0500
 SQL authorization ID   = CRUS
 Local database alias   = DB2SPM
```

2. Issue the LIST DRDA INDOUBT TRANSACTIONS command to display the indoubt transactions known to the SPM. The example below shows one indoubt transaction known to the SPM. The db_name is the local alias for the AS. The partner_lu is the fully qualified luname of the AS. This provides the best identification of the AS and should be provided by the caller from the AS. The luwid provides a unique identifier for a transaction and is available at all DRDA Application Servers. If the transaction in question is displayed, then the uow_status field can be used to determine the outcome of the transaction if the value is C (commit) or R (rollback).

```
db2 => list drda indoubt transactions
 DRDA Indoubt Transactions:
 1.db_name: DBAS3    db_alias: DBAS3    role: AR
   uow_status: C  partner_status: I  partner_lu: USIBMSY.SY12DQA
 corr_tok: USIBMST.STB3327L
 luwid: USIBMST.STB3327.305DFDA5DC00.0001
 xid: 53514C2000000017 00000000544D4442 0000000000305DFD A63055E962000000
      00035F
```

3. If an indoubt transaction for the partner_lu and for the luwid is not displayed, or if the LIST DRDA INDOUBT TRANSACTIONS command returns as follows:

   ```
   db2 => list drda indoubt transactions
   SQL1251W  No data returned for heuristic query.
   ```

   then the transaction was rolled back.

There is another unlikely but possible situation that may occur. If an indoubt transaction with the proper luwid for the partner_lu is displayed, but the uow_status is "I" then the SPM doesn't know whether the transaction is to be committed or rolled back. The XID displayed in the message can be used to determine the proper commit decision as described in the “Manual Recovery of Indoubt Transactions” on page 250 section.

Recovery Using TCP/IP Communications

Recovery of indoubt DRDA transactions involving TCP/IP communications environment has some differences to that for indoubt transactions involving a SNA communications environment. When an indoubt transaction occurs in a TCP/IP communications environment, an alert entry is generated at the client, at the database server, and/or at the the Transaction Manager (TM) database depending on who detected the problem. The alert entry is placed in the `db2alert.log` file. For more information on alerts, see the *Troubleshooting Guide* manual.

The resynchronization of any indoubt transactions occurs automatically as soon as the TM and the participating databases and their connections are all available again.

It is also possible to use the LIST INDOUBT TRANSACTIONS WITH PROMPTING command to support the heuristic operations such as heuristic forget, commit, and roll back. Tools can be written using the Database Monitor GET SNAPSHOT API sqlmonss() and the Transaction Heuristic API sqlxhqry(), sqlxfrg(), sqlxhcom(), and sqlxhrol() to perform these heuristic operations. For more information on these two APIs, see the *System Monitor Guide and Reference* for the first API and the *API Reference* for the second.

Chapter 8. Using DB2 with an XA-Compliant Transaction Manager

You may want to use your databases with an XA-compliant transaction manager if you have resources other than DB2 databases that you want to participate in a two-phase commit transaction. If your transactions only access DB2 databases, you should use the DB2 transaction manager as described in “Updating Multiple Databases” on page 242.

The following topics are provided to assist you in using the database manager with an XA-compliant transaction manager such as CICS for AIX:

In addition, Appendix G, “X/Open Distributed Transaction Processing Model” on page 671 provides concepts about the XA interface and the support provided by DB2. You may wish to review this overview if you are unfamiliar with TP Monitor environments, such as CICS.

If you are using an XA-compliant transaction manager, or are implementing one, please search our technical support web site. The URL to the web site is:
`http://www.software.ibm.com/data/db2/library/`

Once there, choose “DB2 Universal Database.” Then search the web site with keyword “XA” for the latest available information on XA-compliant transaction managers.

Setting Up a Database as a Resource Manager

Each database is defined as a separate resource manager (RM) to the transaction manager (TM), and the database must be identified with an XA open string that has the following syntax:

```
"database_alias<,username,password>"
```

The `database_alias` is required to specify the database alias name of the database. This alias name is the same as the database name unless you have explicitly cataloged an alias name after you created the database. The `username` and `password` are optional, and are used to provide authentication information to the database if the database manager instance is set up with the `authentication` database manager configuration parameter set to SERVER.

When setting up a database as a resource manager, you do not require the XA close string. This string will be ignored by the database manager if it is provided.

A program can access different databases using the SQL CONNECT statement. Each transaction can access one or more databases as described in Chapter 7, "Distributed Databases" on page 239. Every database to be accessed in the transaction must be defined as a resource manager using an XA open string. If a database is not defined as a resource manager, that database cannot be used within a transaction controlled by an XA-compliant transaction manager.

The database manager allows both non-XA and global transactions to access local and remote instances of the database manager. If all the databases reside on machines separated from the TP Monitor machine, the TP Monitor machine uses the database client to forward the XA and SQL requests to the databases.

The database manager also supports access from global transactions to databases that implement the Distributed Relational Database Architecture (DRDA) to ensure data integrity with other RMs accessed by the same transaction. See the *Embedded SQL Programming Guide* for information about the SQL statements that are allowed in this environment.

The access to DRDA databases is dependent on the type of application server:

- DRDA1

  A two-phase commit transaction can only read a database on a DRDA1 application server.

  If read/write accesses are required against DRDA1 databases, the transactions must be run as non-XA transactions. For example, in a CICS for AIX environment, this will require running the transaction from a CICS region that does not define any DB2 for AIX databases in its XAD definition. The SQL COMMIT or ROLLBACK statement has to be explicitly coded in a non-XA application.

- DRDA2

  Two-phase commit is possible with DRDA2 Application Servers provided the DRDA Server is accessed using SNA and using the DB2 Connect Enterprise Edition LU 6.2 Sync Point Manager (SPM). Access to DB2 Connect/SPM can be either local or as a gateway on a remote machine.

  If the SPM is not available, or if the DRDA Server does not support the LU 6.2 syncpoint function, or the DRDA Server is accessed using TCP/IP, then the connection will be read-only, similar to accessing a DRDA1 database.

  Please see *DB2 Connect Enterprise Edition Quick Beginnings* for information regarding the use of the SPM.

Database Connection Considerations

In a partitioned database, database data can be divided into several database partitions. An application accessing this database connects and sends requests to one of the database partitions. Different applications can connect to different database partitions; and the same application can choose different database partitions for different connections.

If a transaction is completed by multiple applications, all applications must connect to the same database partition in a partitioned database while performing part of this transaction. If XREG requests with the same XID are sent to different partitions in a partitioned database, DB2 will treat them as separate transactions, instead of a single transaction.

If a transaction is completed by multiple transactions, it is important that the database stays up until the transaction is completed. For example problems can occur in the following scenario: a transaction involves two applications A and B, with A disconnecting after performing its portion of the work, and before B connects to the database to complete the transaction. If there is no other connection and the database has not been activated (by the ACTIVATE DATABASE command), the database is shut down before connection by application B starts up the database again. This means the transaction started by A will be rolled back. In a non-partitioned database, the problem described in this scenario can be easily avoided by having application B connect to the database before application A disconnects. However, this solution would not work in a partitioned database. In the above scenario, it is possible that the work performed by application A involves several database partitions that application B has not sent (and will not send) any request to. So the database connection made by application B would not prevent the database being shut down on these database partitions. It is recommended that the ACTIVATE DATABASE command be used to prevent such undesirable behavior. See the *Command Reference* for more information on ACTIVATE DATABASE.

Making a Heuristic Decision

An XA-compliant transaction manager uses a two-phase commit process similar to that used by the DB2 transaction manager as described in “Understanding the Two-Phase Commit Process” on page 246. The primary difference between the two environments is that the TP Monitor provides the function of logging and controlling the transaction, instead of the transaction manager database (that is used by the DB2 transaction manager).

Errors similar to those discussed for the DB2 transaction manager (see “Recovering from Problems During Two-Phase Commit” on page 249) can occur when using an XA-compliant transaction manager. Similar to the DB2 transaction manager, an XA-compliant transaction manager will attempt to resynchronize indoubt transactions.

If, for some reason, you cannot wait for the transaction manager to automatically resolve the indoubt transaction, there are some actions you can take to manually resolve the states of indoubt transactions. This manual process is sometimes referred to as “making a heuristic decision.”

The LIST INDOUBT TRANSACTIONS command using the WITH PROMPTING option (or the use of a related set of APIs) allows you to query, commit, and roll back indoubt transactions. In addition, it also allows you to “forget” transactions that have been heuristically committed or rolled back by removing the log records and releasing the log space. For information about the command and APIs, see the *Command Reference* and the *API Reference* manuals.

Note: The LIST INDOUBT TRANSACTIONS command (and APIs) can only be used for *Universal Database* versions of DB2. Other types of resource managers, including those controlled by DRDA2-compliant database managers may have other ways to query indoubt transactions and to make heuristic decisions for their resources.

You should use this command (or APIs) with **extreme caution** and as a last resort. The best solution is to wait for the transaction manager to drive the resynchronization process. You could experience data integrity problems, if you manually commit or roll back a transaction in one of the participating databases, and the opposite action is taken for another of the databases. Recovering from data integrity problems requires you to understand the application logic, the data changed or rolled back, and then to perform a point-in-time recovery of the database, or manually undo/redo the database changes.

If you cannot wait for the transaction manager to initiate the resynchronization process and you must release the resources tied up by an indoubt transaction, then heuristic operations are necessary. This situation could occur if the transaction manager will not be available for an extended period of time to do the resynchronization and the indoubt transaction is tying up resources that are urgently needed. An indoubt transaction ties up the resources that were associated with this transaction before the transaction manager or resource managers became unavailable. For the database manager, these resources include things such as the locks on tables and indexes, log space, and storage taken up by the transaction. Each indoubt transaction also decreases (by one) the maximum number of concurrent transactions that can be handled by the database.

There are no foolproof ways to perform heuristic operations. You can use the following steps as a guideline:

1. Connect to the database for which you require all transactions to be complete.
2. Use the LIST INDOUBT TRANSACTIONS command to display the indoubt transactions. The **xid** represents the global transaction ID and is identical to the **xid** used by the transaction manager and by other resource managers participating in this transaction.
3. For each indoubt transaction, use your knowledge about the application and the operating environment to determine the other participating resource managers.
4. Determine if the transaction manager is available:
   - If the transaction manager is **available**, and the indoubt transaction in a resource manager was caused by the resource manager not being available in the second commit phase, or for an earlier resynchronization process, then you should check the transaction manager's log to determine what action has been taken against the other resource managers. You should then take the same action for the database, that is, using the LIST INDOUBT TRANSACTION command, either heuristically commit or heuristically roll back the transaction in the database.
   - If the transaction manager is **not available**, you will need to use the status of the transaction in the other participating resource managers to determine what action you should take:

- If at least one of the other resource managers has committed the transaction, you should heuristically commit the transaction in all the resource managers. (For *common server* versions of DB2, you can use the LIST INDOUBT TRANSACTIONS command to initiate the heuristic actions.)
- If at least one of the other resource managers has rolled back the transaction, you should heuristically roll back the transaction.
- If the transaction is in “PREPARED” (indoubt) state in all of the participating resource managers, you should heuristically roll back the transaction.
- If one or more of the other resource managers is not available, you should heuristically roll back the transaction.

Do not perform the heuristic forget function unless a heuristically committed or rolled back transaction causes a log full condition, as indicated by the **Logfull** condition in the output of the LIST INDOUBT TRANSACTIONS command. The heuristic forget function releases the log space occupied by this indoubt transaction. The implication is that if a transaction manager eventually performs a resynchronization operation for this indoubt transaction, it could potentially make the wrong decision to commit or roll back other resource managers because there is no log record found for the transaction in this resource manager. In general a "missing" log record implies that the resource manager had rolled back the transaction.

Security Considerations

As mentioned in “Application Program (AP)” on page 671, the TP monitor pre-allocates a set of server processes and runs the transactions from different users under the IDs of the server processes. To the database, each server process appears as a big application that has many units of work, all being run under the same ID associated with the server process.

For example, in an AIX environment using CICS, when a CICS for AIX region is started up, it is associated with the AIX username with which it is defined. All the CICS Application Server processes are also being run under this CICS for AIX "master" ID which is usually defined as "cics". CICS users can invoke CICS transactions under their DCE login ID, and while in CICS, they can also change their ID using the CESN signon transaction. [1]

1

Note that CICS for AIX can also interface with an external security manager to verify the signon ID and password. An administrator can also define which users can run specific CICS programs through the control of the Transaction Definition (TD). (TD in CICS for AIX is equivalent to the combination of Program Control Table (PCT) and Transaction List Table (XLT) in the other CICS family members.)

Several security measures can be used to restrict the usage of CICS by AIX users. A user must first be allowed to run the ***cicsh*** command to gain access to the CICS region. A user who is not defined in the CICS User Definition (UD) with specific security and transaction level keys can only have public level access.

In either case, the end user's ID is not available to the RM. Consequently a CICS Application Process might be running transactions on behalf of many users, but they all appear to the RM as if it is a single program with many units of work from the same "cics" ID. Optionally, you may specify a user ID and password on the XA Open string, and that user ID will be used instead of the "cics" ID to connect to the database.

For static SQL statements, there is not much impact because the binder's privileges not the end user's privileges, are used to access the database. This does mean, however, that the EXECUTE privilege of the database packages must be granted to the server's ID and not the end user's.

For dynamic statements, which have their access authentication done at run-time, this means that the access privileges of the database objects must be granted to the server's ID and not to the actual user of those objects. Instead of relying on the database to control the access of specific users, you must rely on the TP Monitor system to determine which users can run which programs. The server ID must be granted all privileges that its SQL users require.

To determine who has accessed a database table or view, you can perform the following steps:

1. From the SYSCAT.PACKAGEDEP catalog view, obtain a list of all packages that depend on the table or view.
2. Determine the names of the server programs (for example, CICS programs) that correspond to these packages through the naming convention used in your installation.
3. Determine the client programs (for example, CICS transaction IDs) that could invoke these programs, and then use the TP Monitor's log (for example, CICS log) to determine who had run these transactions or programs and at what times.

Configuration Considerations

You should consider the values of the following configuration parameters when you are setting up your TP Monitor environment:

- “Transaction Processor Monitor Name (tp_mon_name)” on page 567

  The *tp_mon_name* database configuration parameter identifies the name of the transaction processor (TP) monitor product being used. For example, "CICS" or "ENCINA".

- “APPC Transaction Program Name (tpname)” on page 547

  The *tpname* database configuration parameter identifies the name of the remote transaction program that the database client must use when issuing an allocate request to the database server when using the APPC communications protocol. This database manager configuration parameter is set in the configuration file at the server and must be the same as the transaction processor (TP) name configured in the SNA transaction program. See the *Quick Beginnings* manuals for more information.

- “Transaction Manager Database Name (tm_database)” on page 533

The *tm_database* database configuration parameter identifies the name of the transaction manager (TM) database for each DB2 instance. A TM database can be a local database or a remote database that is not accessed though DRDA protocols. A TM database can be used as a logger and coordinator, and is used to perform recovery of indoubt transactions.

- "Maximum Number of Active Applications (maxappls)" on page 508

  The *maxappls* database configuration parameter specifies the maximum number of active applications allowed.

  The value of this parameter must be large enough to accommodate the application (user) connections to the database, as well as the number of indoubt units of work and the number of user transactions that have entered, but not yet completed, a two-phase syncpoint process.

  As a result, for a Transaction Processing Monitor environment (for example, CICS for AIX) you may need to increase the value of the *maxappls* parameter. Increasing the value helps ensure that all TP Monitor processes can be accommodated.

- "Auto Restart Enable (autorestart)" on page 529

  This database configuration parameter specifies whether the RESTART DATABASE routine will be invoked when needed. The default is yes (that is, enabled).

  A database containing indoubt transactions will require the RESTART DATABASE command/routine to be invoked in order to start up. If the *autorestart* option is not enabled, when the last connection to the database is dropped, the next connection will fail and require an explicit RESTART DATABASE again. This condition will exist until the indoubt transaction has been removed either by the transaction manager's resync operation, or through a heuristic operation performed by the administrator. When the RESTART DATABASE is issued, a message will be displayed if there are any indoubt transactions in the database. The administrator can then use the LIST INDOUBT TRANSACTION command and other command line processor commands to find out information about those indoubt transactions.

XA Interface Problem Determination

When an error is detected during an XA request from the TM, the application program may not be able to get the error code from the TM. If your program abends or gets a cryptic return code from the TP Monitor or the TM, you should check the First Failure Service Log, which reports XA error information when diagnostic level 3 or greater is in use.

For more information about the First Failure Service Log, see the *Troubleshooting Guide* manual. In addition to this source of information for problem determination, you should also consult the console message, TM error file or other product-specific information provided by the external transaction processing software being used. Refer to the documentation of your transaction processing product for more details in this area.

The database manager writes all XA specific errors to the First Failure Service Log with SQLCODE -998 (transaction or heuristic errors) and the appropriate reason codes. The following are some of the more common reasons for errors:

- Invalid syntax in the XA open string.
- Failure to connect to the database specified in the open string as a result of one of the following:
  - The database has not been cataloged
  - The database has not been started
  - The server application's username/password is not authorized to connect to the database.
- Communications error.

The following example displays an XA open error generated on an AIX platform due to a missing XA open string.

```
Tue Apr  4 15:59:08 1995
toop pid(83378) process (xatest) XA DTP Support      sqlxa_open Probe:101
DIA4701E Database "" could not be opened for distributed transaction
processing.
String Title : XA Interface SQLCA  pid(83378)
SQLCODE = -998  REASON CODE: 4  SUBCODE: 1
Dump File : /u/toop/diagnostics/83378.dmp Data : SQLCA
```

Figure 37. Error Log for XA Open Error

Using Encina for Transaction Processing Through TM-XA Interface

When using Encina for transaction processing with DB2 through the TM-XA interface, please note the following:

1. Encina nested transactions are not currently supported by the DB2 XA interface. If possible, avoid their use. If their use cannot be avoided, make sure that SQL work is done in only one member of the Encina transaction family.
2. If a thread of control is associated with a particular transaction and intends to switch the association to a different transaction, it can do so only after resolving (committing or aborting) the transaction with which it is associated currently.

In the event that a RELEASE statement is used to release a connection to a database, a CONNECT statement, rather than SET CONNECTION, should be used to reconnect to that database.

Part 3. Tuning Application Performance

Chapter 9. Application Considerations

There are a number of factors that can impact the runtime performance of your application. This chapter describes the following topics that should be considered when you are designing and coding your application:

- Concurrency
- Locking
- Adjusting the Optimization Class
- Quickly Retrieving the First Few Rows Using OPTIMIZE FOR n ROWS
- Row Blocking
- Tuning Queries
- Compound SQL
- Performance Considerations and Character Conversion
- Stored Procedures
- Activating a Database
- Parallel Processing of Applications.

You should also refer to the *Embedded SQL Programming Guide* for additional information which can affect the performance of your applications, for example:

- Writing programs using embedded static SQL
- Writing programs using embedded dynamic SQL.

Concurrency

The integrity of the data in a relational database must be maintained as multiple users access and change the data. *Concurrency* is the sharing of resources by multiple interactive users or application programs at the same time. The database manager controls this access to prevent undesirable effects, such as:

- *Lost updates*. Two applications, A and B, might both read the same row from the database and both calculate new values for one of its columns based on the data these applications read. If A updates the row with its new value and B then also updates the row, the update performed by A is lost.
- *Access to uncommitted data*. Application A might update a value in the database, and application B might read that value before it was committed. Then, if the value of A is not later committed, but backed out, the calculations performed by B are based on uncommitted (and presumably invalid) data.
- *Nonrepeatable reads*. Some applications involve the following sequence of events: application A reads a row from the database, then goes on to process other SQL requests. In the meantime, application B either modifies or deletes the row and commits the change. Later, if application A attempts to read the original row again, it receives the modified row or discovers that the original row has been deleted.
- *Phantom Read Phenomenon*. The phantom read phenomenon occurs when:
  1. Your application executes a query that reads a set of rows based on some search criterion.

2. Another application inserts new data or updates existing data that would satisfy your application's query.
3. Your application repeats the query from step 1 (within the same unit of work).

When the query is repeated (step 3), some additional ("phantom") rows are returned as part of the result set that were not returned when the query was initially executed (step 1).

An *isolation level* determines how data is locked or isolated from other processes while the data is being accessed. The isolation level will be in effect for the duration of the unit of work. Applications that use a cursor declared using the WITH HOLD clause will keep the chosen isolation level for the duration of the unit of work in which the OPEN CURSOR was performed. (For more information, refer to the *SQL Reference* manual.) See "Specifying the Isolation Level" on page 269 for information on how the isolation level is specified.

DB2 supports the following isolation levels:

- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read.

(Note that some DRDA database servers support the *no commit* isolation level. On other databases, it behaves like the uncommitted read isolation level. Refer to the *SQL Reference* for information on this isolation level.)

See also:

- "Choosing the Isolation Level" on page 268
- "Specifying the Isolation Level" on page 269.

Repeatable Read

Repeatable read (RR) locks all the rows an application references within a unit of work. Using repeatable read, a SELECT statement issued by an application twice within the same unit of work, in which the cursor was opened, gives the same result each time. With repeatable read, lost updates, access to uncommitted data, and phantom rows are not possible.

The repeatable read application can retrieve and operate on the rows as many times as needed until the unit of work completes. However, no other applications can update, delete, or insert a row that would affect the result table, until the unit of work completes. Repeatable read applications cannot see uncommitted changes of other applications.

With repeatable read, every row that is referenced is locked, not just the rows that are retrieved. Appropriate locking is performed so that another application cannot insert or update a row that would be added to the list of rows referenced by your query, if the query was re-executed. This prevents phantom rows from occurring. This means that if you scan 10 000 rows and apply predicates to them, locks are held on all 10 000 rows, even though only 10 rows qualify.

Note: The repeatable read isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

Since repeatable read may acquire and hold a considerable number of locks, these locks may exceed the number of locks available as a result of the *locklist* and *maxlocks* configuration parameters. (See “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 500 and “Maximum Storage for Lock List (locklist)” on page 477.) In order to avoid lock escalation, the optimizer may elect to immediately acquire a single table level lock for an index scan, if it believes that lock escalation is very likely to occur. (See “Lock Escalation” on page 275 for a discussion of lock escalation.) This functions as though the database manager has issued a LOCK TABLE statement on your behalf. If you do not want a table level lock to be obtained ensure that enough locks are available to the transaction or use the Read Stability isolation level.

Read Stability

Read stability (RS) locks only those rows that an application retrieves within a unit of work. It ensures that any qualifying row read during a unit of work is not changed by other application processes until the unit of work completes, and that any row changed by another application process is not read until the change is committed by that process. That is, “nonrepeatable read” behavior is **not** possible.

Unlike repeatable read, with read stability, if your application issues the same query more than once, you may see additional *phantom* rows (the *phantom read phenomenon*). Recalling the example of scanning 10 000 rows, read stability only locks the rows that qualify. Thus, with read stability, only 10 rows are retrieved, and a lock is held only on those ten rows. Contrast this with repeatable read, where in this example, locks would be held on all 10 000 rows. The locks that are held can be share, next share, update, or exclusive locks. (For more information on lock attributes, see “Attributes of Locks” on page 271.)

Note: The read stability isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

One of the objectives of the read stability isolation level is to provide both a high degree of concurrency as well as a stable view of the data. To assist in achieving this objective, the optimizer ensures that table level locks are not obtained until lock escalation occurs. (See “Lock Escalation” on page 275 for more information about lock escalation).

The read stability isolation level is best for applications that include all of the following:

- Operate in a concurrent environment
- Require qualifying rows to remain stable for the duration of the unit of work
- Do not issue the same query more than once within the unit of work, or do not require that the query get the same answer when issued more than once in the same unit of work.

Cursor Stability

Cursor stability (CS) locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.

No other applications can update or delete a row that a cursor stability application has retrieved while any updatable cursor is positioned on the row. Cursor stability applications cannot see uncommitted changes of other applications.

Recalling the example of scanning 10 000 rows, if you use cursor stability, you will only have a lock on the row under your current cursor position. The lock is removed when you move off that row (unless you update that row).

With cursor stability, both nonrepeatable read and the phantom read phenomenon are possible. Cursor stability is the default isolation level and should be used when you want the maximum concurrency while seeing only committed rows from other applications.

Uncommitted Read

Uncommitted read (UR) allows an application to access uncommitted changes of other transactions. The application also does not lock other applications out of the row it is reading, unless the other application attempts to drop or alter the table. Uncommitted read works differently for read-only and updatable cursors.

Read-only cursors can access most uncommitted changes of other transactions. However, tables, views, and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes by other transactions can be read before they are committed or rolled back.

Cursors that are updatable operating under the uncommitted read isolation level will behave as if the isolation level was cursor stability.

Recalling the example of scanning 10 000 rows, if you use uncommitted read, you do not acquire any row locks.

With uncommitted read, both nonrepeatable read behavior and the phantom read phenomenon are possible.

The uncommitted read isolation level is most commonly used for queries on read-only tables, or if you are only executing select-statements and you do not care whether you see uncommitted data from other applications.

Choosing the Isolation Level

Table 23 on page 269 summarizes the different isolation levels in terms of the undesirable effects described in *Embedded SQL Programming Guide* manual.

Table 23. Summary of isolation levels

Isolation Level	Access to Uncommitted Data	Nonrepeatable Reads	Phantom Read Phenomenon
Repeatable Read (RR)	Not Possible	Not Possible	Not Possible
Read Stability (RS)	Not Possible	Not Possible	Possible
Cursor Stability (CS)	Not Possible	Possible	Possible
Uncommitted Read (UR)	Possible	Possible	Possible

Table 24 provides a simple heuristic that may help you choose an initial isolation level for your applications. Consider this table as a starting point, and refer to the previous discussions of the various levels for factors that might make another value more appropriate for your requirements.

Table 24. Guidelines for choosing an isolation level

Application Type	High data stability required	High data stability not required
Read-write transactions	RS	CS
Read-only transactions	RR	UR

Choosing the appropriate isolation level for an application is very important to avoid the phenomena that are intolerable for that application. The isolation level affects not only the degree of isolation among applications but also the performance characteristics of an individual application since the CPU and memory resources, required to obtain and free locks, vary with the isolation level. The potential for deadlock situations also varies with the isolation level.

Specifying the Isolation Level

The isolation level is specified at precompile time or when an application is bound to a database. For an application written in a supported compiled language, use the ISOLATION option of the command line processor PREP or BIND commands. The isolation level can also be specified by using the PREP or BIND APIs. If no isolation level is specified, the default of cursor stability is used.

If a bind file is created at precompile time, the isolation level is stored in the bind file. If no isolation level is specified at bind time, the default is the isolation level used during precompilation.

You can determine the isolation level of a package by executing the following query:

```
SELECT ISOLATION FROM SYSCAT.PACKAGES
  WHERE PKGNAME = 'XXXXXXXX'
  AND PKGSCHEMA = 'YYYYYYYY'
```

where *XXXXXXXX* is the name of the package and *YYYYYYYY* is the schema name of the package. Both of these names must be in all capital letters.

When a database is created, multiple bind files used to support the different isolation levels for SQL in REXX are bound to the database (on those servers that support REXX). Refer to the *Embedded SQL Programming Guide* for more information about bind files.

REXX and the command line processor connect to a database using a default isolation level of cursor stability. Changing to a different isolation level does not change the connection state. It must be executed in the CONNECTABLE AND UNCONNECTED state or in the IMPLICITLY CONNECTABLE state. (See the CONNECT TO statement in the *SQL Reference* for details about connection states.) You cannot be connected to a database when issuing this command.

The isolation level being used can be checked by a REXX application by checking the value of the SQLISL REXX variable. The value is updated every time the CHANGE SQLISL command is executed.

If you are using the command line processor you may change the isolation level using the CHANGE ISOLATION LEVEL command. Refer to the *Command Reference* manual for more information.

For DB2 Call Level Interface (DB2 CLI), you may change the isolation level as part of the DB2 CLI configuration. In addition, many commercially-written applications also provide a method to allow you to choose the isolation level. Refer to the *CLI Guide and Reference* manual for more information.

Locking

The database manager provides concurrency control and prevents uncontrolled access by means of locks. A *lock* is a means of associating a database manager resource with an application to control how other applications can access the same resource. The application with which the resource is associated is said to hold or own the lock.

The database manager imposes locks to prohibit applications from accessing uncommitted data written by other applications (unless the uncommitted read isolation level is used). This principle protects data integrity (that is, the consistency and security of data). Locks can also prohibit the updating of rows (such as for a repeatable read application).

To satisfy data integrity, the database manager acquires locks implicitly, under database manager control. Except for the uncommitted read isolation level, it is never necessary for an application to request a lock explicitly to ensure that uncommitted data is hidden from other processes.

Because of the basic principle of locking, you do not need to take action to control locks in most cases. Still, applications acquire locks on the basis of certain general parameters. Knowledge of your local situation can help you make better use of your system resources by changing those parameters. To assist you, the following topics on locking are discussed:

- Attributes of Locks
- Locks and Application Performance
- Factors Affecting Locking
- LOCK TABLE Statement
- CLOSE CURSOR WITH RELEASE
- Summary of Locking Considerations

Attributes of Locks

Database manager locks have the following basic attributes:

Object The resource being locked. The only types of explicitly lockable objects are tables. The database manager also imposes locks on other types of resources, such as rows, tables and table spaces. The object being locked represents the *granularity* of the lock.

Duration The length of time a lock is held. Lock durations are affected by isolation levels which are discussed in "Concurrency" on page 265.

Mode The type of access allowed for the lock owner as well as the type of access permitted for concurrent users of the locked object. It is sometimes referred to as the *state* of the lock.

Modes and their effects are shown in order of increasing control over resources:

IN (Intent None)
The lock owner can read any data in the table, including uncommitted data, but cannot change any of it. No row locks are acquired by the lock owner. Other concurrent applications can read or update the table. Both table spaces and tables can be locked in this mode.

IS (Intent Share)
The lock owner can read data in the locked table, but not change this data. When an application holds the IS table lock, the application acquires an S or NS lock on each row read. In either case, other applications can read or update the table. Both table spaces and tables can be locked in this mode.

NS (Next Key Share)
This lock is acquired on rows of a table, instead of a Share lock. The lock owner and all concurrent applications can read, but not change, the locked row. Only individual rows can be locked in NS mode. This lock is acquired in place of a share (S) lock on data that is read with the RS or CS isolation levels.

S (Share)
The lock owner and any concurrent applications can read, but not change, the locked data. Individual rows can be Share locked. If a table is Share locked, no row locks are acquired by the lock owner. Other concurrent applications can read the table. Both rows and tables can be locked in this mode.

IX (Intent Exclusive)

The lock owner and concurrent applications can read and change data in the table. When the owner reads data, it acquires an S, NS, X, or U lock on each row. It also acquires an X lock on each row that it updates. Other concurrent applications can both read and update the table. Both table spaces and tables can be locked in this mode.

SIX (Share with Intent Exclusive)

The lock owner can both read and change data in the table. The lock owner acquires X locks on the rows it updates, but does not acquire locks on rows that it reads. Other concurrent applications can read the table. Only a table object can be locked in this mode.

U (Update)

The lock owner can update data in the locked object and acquire X locks on the rows prior to updates. Other units of work can read the data, but cannot attempt to update it. Both rows and tables can be locked in this mode. When a table is locked in U mode, X row locks are obtained.

NX (Next Key Exclusive)

This lock is acquired on the next row when a row is deleted from an index or inserted into the index of a table. The lock owner can read but not change the locked row. Only individual rows can be locked in NX mode. This is similar to an X lock except that it is compatible with the NS lock.

NW (Next Key Weak Exclusive)

This lock is acquired on the next row when a row is inserted into the index of a non-catalog table. The lock owner can read but not change the locked row. Only individual rows can be locked in NW mode. This is similar to X and NX locks except that it is compatible with the W and NS locks.

X (Exclusive)

The lock owner can both read and change data in the locked object. Tables can be Exclusive locked, meaning that no row locks will be acquired. Only uncommitted read applications can access the locked table. Both rows and tables can be locked in this mode.

W (Weak Exclusive)

This lock is acquired on the row when a row is inserted into a non-catalog table. The lock owner can change the locked row. Only individual rows are locked in W mode. This lock is similar to an X lock except that it is compatible with the NW lock. Only uncommitted read applications can access the locked row.

Z (Superxclusive)

This lock is acquired on a table in certain conditions, such as when the table is altered or dropped, an index on the table is created or dropped, or a table is reorganized. No other concurrent application can read or update the table. Both table space and table objects can be locked in this mode.

Note that only tables and table spaces will obtain the “intent” lock modes. That is, intent locks are not obtained for rows.

Locks and Application Performance

Application programmers need to be aware of several related factors concerning the uses of locks and their effect on the performance of applications. These factors include the following:

- Concurrency and Granularity
- Lock Compatibility
- Lock Conversion
- Lock Escalation
- Lock Waits and Timeouts
- Deadlocks.

Concurrency and Granularity

A lock held by one application can prevent access by another application. Therefore, for maximum concurrency, a row level lock is better than a table lock. But locks require storage and processing time to manage. Therefore, for minimizing storage and processing time, one table lock is better than many row locks.

Lock Compatibility

Table 25 indicates whether a lock request is granted if another process holds or is requesting a lock on the same resource in a given state. A **no** indicates that the requestor must wait until all incompatible locks are released by other processes. Note that a timeout can occur when waiting for a lock. A **yes** indicates that the lock is granted (unless someone else is waiting for the resource).

Table 25. Lock Type Compatibility

	State of Held Resource												
State Being Requested	**none**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**NX**	**X**	**Z**	**NW**	**W**
none	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IN	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
IS	yes	yes	yes	yes	yes	yes	yes	yes	no	no	no	no	no
NS	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes	no
S	yes	yes	yes	yes	yes	no	no	yes	no	no	no	no	no
IX	yes	yes	yes	no	no	yes	no	no	no	no	no	no	no
SIX	yes	yes	yes	no	no	no	no	no	no	no	no	no	no
U	yes	yes	yes	yes	yes	no	no	no	no	no	no	no	no
NX	yes	yes	no	yes	no	no	no	no	no	no	no	no	no
X	yes	yes	no	no	no	no	no	no	no	no	no	no	no
Z	yes	no	no	no	no	no	no	no	no	no	no	no	no
NW	yes	yes	no	yes	no	no	no	no	no	no	no	no	yes
W	yes	yes	no	no	no	no	no	no	no	no	no	yes	no

Abbreviations:

I	Intent
N	None
NS	Next Key Share
S	Share
NX	Next Key Exclusive
X	Exclusive
U	Update
Z	Super Exclusive
NW	Next Key Weak Exclusive
W	Weak Exclusive

For details of these lock types, refer to the discussion in “Attributes of Locks” on page 271.

Legend:

- yes - **grant** lock requested immediately
- no - **wait** for held lock to be released or timeout to occur

Assume that application A holds a lock on a table that application B also wants to access. The database manager requests, on behalf of application B, a lock of some particular mode. If the mode of the lock held by A permits the lock requested by B, the two locks (or modes) are said to be compatible.

If the lock mode requested for application B is not compatible with the lock held by application A, application B cannot continue. Instead, it must wait not only until application A releases its lock, but until *all* existing incompatible locks are released.

Lock Conversion

Lock conversion occurs when a process accesses a data object on which it already holds a lock, and the mode of access requires a more restrictive lock than the one already held. A process can hold only one lock on a data object at any time, although it can (indirectly through a query) request a lock many times on the same data object. The operation of changing the mode of the lock already held is called a *conversion*.

The conversion case for rows is simple: As an example, a conversion occurs if an X is needed and an S or U is held.

There are more distinct lock modes for tables than for rows. IX (Intent Exclusive) and S (Shared) locks are special cases, however. Neither S nor IX is considered to be more restrictive than the other, so if one of these is held and the other required, the resulting conversion is to a SIX (Share with Intent Exclusive) lock. All other conversions result in the requested lock mode becoming the mode of the lock held, if the requested mode is more restrictive.

A query to update a row can also produce a dual conversion. Suppose the row had been read through an index access and was locked as S. The table containing the row would have a covering intention lock. Suppose it is an IS rather than an IX. Then, if the row is subsequently changed, the table lock is converted to an IX, and the row to an X.

As a reminder, the application of locks usually takes place implicitly during the execution of a query. Understanding the kinds of locks obtained for different queries and table and index combinations can assist you in designing and tuning your application. See “Factors Affecting Locking” on page 278 for more information on this topic.

Lock Escalation

Lock escalation occurs when too many locks (of any type) are currently held.

Lock escalation can occur for a specific database agent if the agent exceeds its allocation of the lock list (see “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 500).

Such escalation is handled internally; the only externally detectable result might be a reduction in concurrent access on one or more tables. Normally, in a properly configured database, lock escalation occurs infrequently.

An example of lock escalation is when an application designer uses an index on a large table to increase performance and concurrency; however, the application accesses a large percentage of records in the table. The database manager is not able to predict (in this case) that so much of the table will be locked, and locks each record individually rather than only locking the table either S or X.

Sometimes, the process receiving the escalation request (internally) holds few or no record locks on any table. The reason for this escalation is that one process (or processes) can be holding many locks (although this amount is below the database configuration parameter of locks per process) but not quite enough to trigger the

escalation request. The process might not request another lock or access the database again except to end the transaction. Then another process can request the lock or locks that trigger the escalation request.

If lock escalation reduces concurrency to an unacceptable level, you can do the following:

- Increase the number of locks allowed by increasing the value of the *maxlocks* and/or the *locklist* parameters in the database configuration file. (See “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 500 and “Maximum Storage for Lock List (locklist)” on page 477.) This might be the choice if concurrent access to the table by other processes is most important. However, the overhead of obtaining record level locks can induce more delay to other processes than is saved by concurrent access to a table. (When changing these parameters in a partitioned database, ensure that the parameters are updated on all partitions).
- Locate and adjust the offending process (or processes), which may or may not be the one escalating or rolling back, and issue LOCK TABLE statements explicitly.
- Change the degree of isolation. Note that this may lead to decreased concurrency or reduced isolation.
- Increase the frequency of commits. This tends to reduce the number of locks in existence at a given time. For more information about isolation levels and concurrency, see “Concurrency” on page 265.

Lock Waits and Timeouts

Without lock timeout detection, in an abnormal situation, your application may have to wait for a lock to be released. This might occur, for example, when a transaction is waiting for a lock held by another user's application, and the other user has left their workstation without performing some interaction to allow their application to commit their transaction which would release the lock. Obviously, this results in poorer application performance. To avoid *stalling* your program in such a case, you can use the *locktimeout* configuration parameter to set the maximum time that any application waits to obtain a lock. (See “Lock Timeout (locktimeout)” on page 501.)

Using this parameter helps avoid global deadlocks, especially in distributed unit of work (DUOW) applications. If the lock times out, that is, if the time that the lock request is pending is greater than the *locktimeout* value, your application receives an error and your transaction is rolled back. For example, if *program1* tries to acquire a lock which is already held by *program2*, *program1* returns SQLCODE -911 (SQLSTATE 40001) with reason code 68 if the timeout is expired.

Deadlocks

In the database manager, contention for locks by processes using the database can result in deadlocks. For example, Process 1 locks table A in X (exclusive) mode and Process 2 locks table B in X mode; if Process 1 then tries to lock table B in X mode and Process 2 tries to lock table A in X mode, the processes will be in a deadlock. In a deadlock, both processes are suspended until their second lock request is granted, and neither request is granted until one of the processes performs a commit or rollback.

This state remains indefinitely until an outside agent activates one of the processes and forces it to perform a rollback.

Deadlocks in the lock system are handled in the database manager by an asynchronous system background process called the deadlock detector. The deadlock detector becomes active periodically as determined by the *dlchktime* configuration parameter (see "Time Interval for Checking Deadlock (dlchktime)" on page 499). When the deadlock detector becomes active, it examines the lock system for deadlocks. If the database has been partitioned then each partition sends *lock graphs* to the catalog node where global deadlock detection takes place.

If a deadlock is found, the deadlock detector selects a deadlocked process to roll back. The selected process is awakened, and it returns to the calling application with SQLCODE -911 (SQLSTATE 40001), with reason code 2. The database manager rolls back the selected process automatically. When the rollback has completed, the locks belonging to the victim process are released, and the other processes involved in the deadlock can eventually proceed.

Selecting the proper interval for the deadlock detector is necessary to ensure good performance. An interval that is too short would cause unnecessary overhead, and one that is too long would allow a deadlock to delay a process for an unacceptable amount of time. For example, a wakeup interval set to 30 minutes could allow a deadlock to exist for nearly 30 minutes. The application designer must balance the possible delays in resolving deadlocks with the overhead of detecting them.

In a partitioned database, the interval should be the same on all partitions (the *dlchktime* configuration parameter must be updated to the same value on all partitions). If the value is smaller at the catalog node than at other partitions, phantom deadlocks may be detected. If the value is larger at the catalog node than at other partitions, it may appear as if more than two intervals pass before a deadlock is detected. If a large number of deadlocks are detected in a partitioned database, you should increase the value of the *dlchktime* parameter to account for lock waits and communication waits.

Another problem can occur when an application with more than one independent process accessing the database is structured in such a way as to make deadlocks likely. An example is an application in which several processes access the same table for reads and then writes. If the processes do read-only SQL queries at first and then do SQL updates on the same table, the chances of deadlocks occurring increase because of potential contention between the processes for the same data. For instance, if two processes read the table, and then update the table, they get into a state where process A is trying to get an X lock on a row, on which process B has an S lock and vice versa. The result could be a deadlock. To avoid these deadlocks, applications that access data with the intention of modifying it should use the FOR UPDATE OF clause when performing a select. This clause ensures that a U lock is imposed when process A attempts to read the data.

Factors Affecting Locking

The mode and granularity of database manager locks are determined by a combination of factors: the type of processing the application performs, how it accesses data, and several parameters that you can specify.

Application Processing

For the purpose of determining lock attributes, processing can be classified as one of four types:

Read-only This type includes all select-statements which are intrinsically read-only (refer to the *SQL Reference* for information about cursors), have an explicit FOR READ ONLY clause, or are ambiguous but for which the SQL compiler presumes to be read-only due to the value of the BLOCKING option specified on the PREP or BIND command. It requires only Share locks (S or IS).

Intent to change This type includes all select-statements with the FOR UPDATE clause, or which the SQL compiler presumes to be intended for change as a result of the interpretation of the ambiguous statement. It uses Share and Update locks (S, U, and X for rows, IX, U, X for tables).

Change This type includes UPDATE, INSERT, and DELETE, but not UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. It requires Exclusive locks (X or IX).

Cursor controlled This type includes UPDATE WHERE CURRENT OF and DELETE WHERE CURRENT OF. It also requires Exclusive locks (X or IX).

A statement that inserts, updates or deletes against a target table, based on the result from a sub-select statement, does two types of processing. The locks for the tables returned in the sub-select are determined by the rules for read-only processing; for the target table, by the rules for change processing.

Access Paths

An *access path* is the method selected by the optimizer for retrieving data from a specific table reference. (See “Data Access Concepts and Optimization” on page 349.) The access path chosen by the optimizer can have a significant effect on the lock modes. For example, when an index scan is used to locate a specific row, the optimizer will likely choose row-level locking (IS) for the table. This type of access would be used to select information for a single employee from the EMPLOYEE table, that has an index on employee number (EMPNO), with a statement such as the following:

```
SELECT *
  FROM EMPLOYEE
  WHERE EMPNO = '000310';
```

Similarly, when no index is used, the entire table must be scanned in sequence to find the selected rows, and may acquire a single table level lock (S). For example, this type

of access might be used to select all the male employees, using a statement such as this:

```
SELECT *
  FROM EMPLOYEE
  WHERE SEX = 'M';
```

The following tables provide an overview of which locks are obtained for what kind of access plan. See “Application Processing” on page 278 for definitions of the column headings. Also see “Data Access Concepts and Optimization” on page 349 for definitions of the access method. Note that *cursor controlled* type processing uses the lock mode of the underlying cursor until the application finds a row to update or delete. For this type of processing, no matter what the lock mode of a cursor, an exclusive lock will always be obtained to perform the update or delete.

In the following tables, if only one lock mode is shown, it is a table level lock mode. If two lock modes are shown, the first is the table level lock mode and the second is the row level lock mode.

Table 26. Lock Modes for Table Scans

Isolation Level	Read-only	Intent to Change	Change
Access Method: *Table scan with no predicates*			
RR	S	U	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Table Scan with predicates*			
RR	S	U	U
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

Table 27 (Page 1 of 2). Lock Modes for Index Scans

Isolation Level	Read-only	Intent to Change	Change
Access Method: *Index Scan with no predicates*			
RR	S	IX / U	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Index Scan a single qualifying row*			
RR	IS / S	IX / U	IX / X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X

Table 27 (Page 2 of 2). Lock Modes for Index Scans

Isolation Level	Read-only	Intent to Change	Change
UR	IN	IX / U	IX / X
Access Method: *Index Scan with start and stop predicates only*			
RR	IS / S	IX / S	IX / X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Index Scan with predicates*			
RR	IS / S	IX / S	IX / U
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

Table 28 shows the lock modes for cases in which reading of the data pages is deferred to allow the list of rows to be:

- Further qualified using multiple indexes. See “Multiple Index Access” on page 354 for more information.
- Sorted for efficient prefetching. See “Understanding List Prefetching” on page 407 for more information.

The deferred access of the data pages implies that access to the row occurs in two steps and this results in more complex locking scenarios. There are two major categories which depend on the isolation level. Since the repeatable read isolation level keeps all locks acquired until the end of the transaction, the locks acquired in the first step are held and there is no need to acquire further locks in the second step. For the read stability and cursor stability isolation levels, locks must be acquired during the second step. To maximize concurrency, we don't acquire locks during the first step and rely on the re-application of all predicates to ensure that only qualifying rows are returned.

Table 28 (Page 1 of 2). Lock Modes for Index Scans used for Deferred Data Page Access

Isolation Level	Read-only	Intent to Change	Change
Access Method: *Index Scan with no predicates*			
RR	IS / S	IX / S	X
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
Access Method: *Deferred Data Page Access, after an index scan with no predicates*			
RR	IN	IX / S	X
RS	IS / NS	IX / U	IX / X

Table 28 (Page 2 of 2). Lock Modes for Index Scans used for Deferred Data Page Access

Isolation Level	Read-only	Intent to Change	Change
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: ***Index Scan with predicates***			
RR	IS / S	IX / S	IX / S
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
Access Method: ***Index Scan with start and stop predicates only***			
RR	IS / S	IX / S	IX / X
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
Access Method: ***Deferred Data Page Access, after an index scan with predicates***			
RR	IN	IX / S	IX / S
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

The access path is not controlled by the user; it is chosen by the Optimizer.

The access path used can affect the mode and granularity of a lock. For example, in an application using the repeatable read (RR) isolation level, an UPDATE query that uses a table scan without predicates, would use an X lock on the table. If rows were located through an index, the database manager might choose to lock individual rows of the table.

LOCK TABLE Statement

You can override the rules for acquiring initial lock modes by using the LOCK TABLE statement in an application.

The statement locks an entire table. Only the table specified in the LOCK TABLE statement is locked. Parent and dependent tables of the specified table are not locked. You must determine whether locking other tables that can be accessed is necessary to achieve the desired result in terms of concurrency and performance. The lock is not released until the unit of work is committed or rolled back.

If a table is normally shared among several users, you might want to lock it for the following reasons:

LOCK TABLE IN SHARE MODE

You want to access data that is *consistent in time*; that is, data current for a table at a specific point in time. If the table experiences frequent activity, the only way to ensure that the entire table remains stable is to lock it. For example, your application wants to take a snapshot of a table. However, during the time your application needs to process some rows of a table, other applications are updating rows you have not yet processed. This is allowed with repeatable read, but this action is not what you want.

As an alternative, your application can issue the LOCK TABLE IN SHARE MODE statement: no rows can be changed, regardless of whether you have retrieved them or not. You can then retrieve as many rows as you need, knowing that the rows you have retrieved have not been changed just before you retrieved them.

With LOCK TABLE IN SHARE MODE, other users can retrieve data from the table, but they cannot update, delete, or insert rows into the table.

LOCK TABLE IN EXCLUSIVE MODE

You want to update a large part of the table. It is less expensive and more efficient to prevent all other users from accessing the table than it is to lock each row as it is updated, and then unlock the row later when all changes are committed.

With LOCK TABLE IN EXCLUSIVE MODE, all other users are locked out; no other applications can access the table unless they are uncommitted read applications.

For more details on the LOCK TABLE statement, refer to the *SQL Reference* manual.

CLOSE CURSOR WITH RELEASE

When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, all read locks (if any) that have been held for the cursor are released. Read locks are IS, S, and U table locks as well as S, NS, and U row locks. For more information on lock modes, see “Attributes of Locks” on page 271.

The WITH RELEASE clause has no effect for cursors that are operating under the CS or UR isolation levels. When specified for cursors that are operating under the RS or RR isolation levels, the WITH RELEASE clause ends some of the guarantees of those isolation levels. Specifically, an RS cursor may experience the *nonrepeatable read* phenomenon, and an RR cursor may experience either the *nonrepeatable read* or *phantom read* phenomenon.

If a cursor that is originally RR or RS is reopened after being closed using the WITH RELEASE clause, then new read locks will be acquired.

Summary of Locking Considerations

The following are points to remember about locking:

- Small units of work (frequent COMMIT statements) promote concurrent access of data by many users. Include COMMIT statements when your application is logically

at a point of consistency; that is, when the data you have changed is consistent. When a COMMIT is issued, locks are released (except for table locks associated with cursors declared WITH HOLD).

- Locks are acquired even if your application merely reads rows, so it is still important to commit read-only units of work. This is because shared locks are acquired by repeatable read, read stability, and cursor stability isolation levels in read-only applications. With repeatable read and read stability, all locks are held until a COMMIT is issued, preventing other processes from updating the locked data, unless you close your cursor using the WITH RELEASE clause. In addition, catalog locks are acquired even in uncommitted read applications using dynamic SQL.
- The database manager ensures that your application does not retrieve uncommitted data (rows that have been updated by other applications but are not yet committed) unless you are using the uncommitted read isolation level.
- You can lock the entire table that you want to protect by issuing a LOCK TABLE statement:
  - To allow other applications to retrieve, but not update, delete, or insert rows
  - To prevent other applications (other than those with an uncommitted read isolation level) from accessing the rows of a table.
- When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, all read locks (if any) that have been held for the cursor are released.
- When changing the configuration parameters affecting locking in a partitioned database, ensure that the changes are made to all of the partitions in the database.

Adjusting the Optimization Class

When an SQL query is compiled, a number of optimization techniques can be used to determine the most efficient access plan for that query. Using more optimization techniques results in:

1. Improvements in run-time performance
2. Increased query compilation time
3. Increased system resource usage.

For this reason, you may wish to limit the number of techniques applied to optimizing your query by setting the optimization class. This can be particularly useful if you have:

- Very small databases or very simple dynamic queries
- Limited memory available at compile time on your database server
- A desire to reduce the query compilation (for example, PREPARE) time.

You may select from any of the query optimization classes described below, although class 0 and class 9 should be used only in special circumstances. Class 5 is the default. Classes 0, 1, and 2 use the Greedy join enumeration algorithm; for complex

queries this algorithm considers far fewer alternative plans, and incurs significantly less compilation time, than classes 3 and above. Classes 3 and above use the Dynamic Programming join enumeration algorithm; this algorithm considers far more alternative plans, and can incur significantly more compilation time, than classes 0, 1, and 2 as the number of tables increases.

0 - This class directs the optimizer to use a minimal amount of optimization to generate an access plan. For example:

- Any non-uniform distribution statistics are not considered by the optimizer.
- Only basic query rewrite rules are applied (see “Query Rewrite by the SQL Compiler” on page 342 for information about query rewrite).
- Greedy join enumeration occurs (see “Search Strategies for Selecting Optimal Join” on page 363).
- Only nested loop join and index scan access methods are enabled (see “Join Concepts” on page 359 and “Index Scan Concepts” on page 349).
- List prefetch and index ANDing are disabled as access methods.
- The star join strategy is not considered.

This class should only be used in special circumstances requiring the lowest possible query compilation overhead. An application consisting entirely of very simple dynamic SQL statements which access well-indexed tables is a good example of where query optimization class 0 is appropriate.

1 - This class directs the optimizer to use a degree of optimization which is roughly comparable to DB2/6000 Version 1, plus some additional low cost features not found in Version 1. In particular:

- Any non-uniform distribution statistics are not considered by the optimizer.
- Only a subset of the query rewrite rules are applied, including those provided in DB2/6000 Version 1.
- Greedy join enumeration (see “Search Strategies for Selecting Optimal Join” on page 363.)
- List prefetch and index ANDing are disabled as access methods.

Optimization class 1 is quite similar to class 0 except that Merge Scan joins and table scans are also available.

2 - This class directs the optimizer to use a degree of optimization which significantly improves upon that of class 1, while keeping the compilation cost significantly lower than classes 3 and above for complex queries. In particular:

- All available statistics, including both frequency and quantile non-uniform distribution statistics, are utilized.
- All of the query rewrite rules are applied, except computationally intensive rules which are applicable only in very rare cases.
- Greedy join enumeration (see “Search Strategies for Selecting Optimal Join” on page 363) is used.
- A wide range of access methods are considered, including list prefetch.
- The star join strategy is considered, if applicable.

Optimization class 2 is quite similar to class 5 except that it uses Greedy join enumeration rather than Dynamic Programming. This class has the most

optimization of all the optimization classes that use the Greedy join enumeration algorithm, which considers fewer alternatives for complex queries, and therefore consumes less compilation time than classes 3 and above. It is therefore recommended for very complex queries in a decision support or on-line analytic processing (OLAP) environment. In such cases, there is a good chance the same query is executed infrequently, so that its access plan is unlikely to remain in the cache until the next occurence of the query.

3 - This class requests that a moderate amount of optimization be performed to generate an access plan. This class comes closest to matching the query optimization characteristics of DB2 for MVS/ESA or OS/390. This optimization class has the following characteristics:

- Non-uniform distribution statistics, which track frequently occurring values are used, if available.
- Most query rewrite rules, including subquery-to-join transformations are applied.
- Dynamic programming join enumeration (see “Search Strategies for Selecting Optimal Join” on page 363):
  - Limited use of composite inner tables (see “Composite Tables” on page 364)
  - Limited use of Cartesian products for star schemas involving “look-up” tables (see “Search Strategies for Star Join” on page 363)
- A wide range of access methods are considered, including list prefetch and index ANDing.

This class is suitable for a broad range of applications. Using this class gives the optimizer a better chance of selecting an excellent access plan for queries with four or more joins. However, the optimizer might fail to consider a better plan which would be chosen with the default query optimization class.

5 - This class directs the optimizer to use a significant amount of optimization to generate an access plan. For example, class 5 has the following characteristics:

- All available statistics including both frequency and quantile non-uniform distribution statistics.
- All of the query rewrite rules are applied, except computationally intensive rules which are applicable only in very rare cases.
- Dynamic programming join enumeration (see “Search Strategies for Selecting Optimal Join” on page 363):
  - Limited use of composite inner tables (see “Composite Tables” on page 364)
  - Limited use of Cartesian products for star schemas involving “look-up” tables (see “Search Strategies for Star Join” on page 363)
- A wide range of access methods are considered, including list prefetch and index ANDing.

When the optimizer detects that the additional resources and processing time are not warranted for complex dynamic SQL queries, optimization is reduced. The extent or size of the reduction is dependent on the machine size and the number of predicates.

When the query optimizer reduces the amount of query optimization performed, it continues to apply all the query rewrite rules that would normally be applied. However, it does use the greedy join enumeration method and reduces the number of access plan combinations that are considered.

Query optimization class 5 is an excellent choice for a mixed environment consisting of both transactions and complex queries. This optimization class has been designed to apply the most valuable query transformations and other query optimization techniques in an efficient manner.

7 - This class directs the optimizer to use a significant amount of optimization to generate an access plan. It is the same as query optimization class 5 except that it does not reduce the amount of query optimization for complex dynamic SQL queries.

9 - This class directs the optimizer to use all available optimization techniques. These include:

- All available statistics
- All query rewrite rules
- All possibilities for join enumerations, including Cartesian products and unlimited composite inners
- All access methods.

This class can greatly expand the number of possible access plans that are considered by the optimizer. This class should be used to determine whether more comprehensive optimization can generate a better access plan for very complex and very long-running queries using large tables. Explain and performance measurements should be used to verify that a better plan has been found.

How Do You Set the Optimization Class?

The way to request a specific query optimization class depends on whether you are using static or dynamic SQL.

- *Static SQL statements* use the optimization class specified on the PREP and BIND commands. The QUERYOPT column in the SYSCAT.PACKAGES catalog table records the optimization class used to bind the package. If the package is rebound either implicitly or using the REBIND PACKAGE command, this same optimization class will be used for the static SQL statements. If you want to change the optimization class used for these static SQL statements, you must use the BIND command. If you do not specify the optimization class, DB2 uses the default optimization as specified by *dft_queryopt*.

- *Dynamic SQL statements* use the optimization class specified by the CURRENT QUERY OPTIMIZATION special register which is set using the SQL SET statement. For example, the following statement sets the optimization class to 1:

  ```
  SET CURRENT QUERY OPTIMIZATION = 1
  ```

  To ensure that a dynamic SQL statement always uses the same optimization class, you may want to include this SET statement in your application program. For more information, refer to the *SQL Reference*.

  If the CURRENT QUERY OPTIMIZATION register has not been set, dynamic statements will be bound using the default query optimization class. The default value for both dynamic and static SQL is determined by value of the configurable database parameter DFT_QUERYOPT. Class 5 is the default query optimization class unless you have changed the default. (For more information on this parameter, see "Default Query Optimization Class (dft_queryopt)" on page 543.) The default values for the bind option and the special register are taken from the DFT_QUERYOPT configuration parameter.

How Much Optimization is Necessary?

Most statements will be adequately optimized using a reasonable amount of resources with the default query optimization class. The query compilation time and resource consumption, at a given optimization class, is primarily influenced by the complexity of the query, particularly the number of joins and subqueries. However, compilation time and resource usage are also affected by the amount of optimization performed for the various optimization classes. For any optimization class, you can expect to see a greater difference in query compilation time and resource consumption for a very complex query than for a simple one.

The following may help you select which optimization class to use:

- Start by using the default query optimization class.
- If you wish to use a class other than the default, try class 1, 2 or 3 first.
- Use a low optimization class (0 or 1) for queries having very short run-times, that is, queries taking less than one second. (See the following discussion for additional criteria about when to choose a low optimization class.)
- Use optimization class 1 or 2 if you have many tables with many of the join predicates that are on the same column, and if compilation time is a concern.
- Use a higher optimization class (3, 5, or 7) for long running queries, that is, queries taking more than 30 seconds.
- Under normal circumstances, you should not use optimization class 9.
- For queries that run a long time, run the query using db2batch to determine how much of the time is spent in compilation and how much is spent in execution.
  - If most of the time is spent in compilation then reduce the optimization class.
  - If most of the time is spent in execution then consider a higher optimization class.

Note that query optimization classes 1, 2, 3, 5, and 7 are all suitable for general purpose use.

Only if you require further reductions in query compilation time and you know the kind of SQL (for example, extremely simple statements) that will be executed should you consider class 0. This SQL will tend to have the following characteristics:

- Access to a single or only a few tables
- Fetches a single or only a few rows
- Uses fully qualified, unique indexes.

Online transaction processing (OLTP) transactions are good examples of this kind of SQL.

Complex queries may require different amounts of optimization to select the best access plan. You may wish to consider using higher optimization classes for queries exhibiting the following characteristics:

- Access to large tables
- A large number of predicates
- Many subqueries
- Many joins
- Many set operators, such as UNION and INTERSECT
- Many qualifying rows
- GROUP BY and HAVING operations
- Nested table expressions
- A large number of views.

Decision support queries or month-end reporting queries against fully normalized databases are good examples of complex queries where at least the default query optimization class should be used.

Another reason to use higher query optimization classes is SQL which was produced by a query generator. Many query generators create SQL which is not efficient. Poorly written queries, including those produced by a query generator, may require additional optimization to make it possible to select a good access plan. Using query optimization class 2 and higher can improve poorly written SQL queries.

The use of static or dynamic SQL, and whether the same dynamic SQL is repeatedly executed are also important considerations. For static SQL, the query compilation time and resources are expended just once and the resulting plan can be used many times. In general, static SQL should always use the default query optimization class. Dynamic statements are bound and executed at run time; therefore, you should consider whether the overhead of additional optimization for dynamic statements improves your overall performance. However, if the same dynamic SQL statement is executed repeatedly, the selected access plan will be cached. For the purposes of selecting a query optimization class, the statement can be treated like a static SQL statement.

(Refer to the *Embedded SQL Programming Guide* for more information about when to use static and dynamic SQL,)

If you think you have a query that could benefit from additional optimization, but you are not sure, or you are concerned about compilation time and resource usage, you may want to perform some benchmark testing. This testing can help you quantify the benefits obtained from different optimization classes. See Chapter 18, "Benchmark Testing" on page 447 for general techniques and the specific use of the `db2batch` tool. When designing and running your benchmark test, consider whether the SQL statements in your application are static or dynamic:

- For **dynamic** SQL statements, your testing should compare the average run time for the statement. You can use the following formula to help you calculate the average run time:

  ```
  compile time + sum of execution times for all iterations
  --------------------------------------------------------
                number of iterations
  ```

  where, the `number of iterations` represents the number of times that you expect that the SQL statement will be executed each time it is compiled.

  Note: Following the initial compilation, dynamic SQL statements are recompiled when a change to the environment requires the statement to be recompiled. Once cached, a SQL statement does not need to be compiled again since subsequent PREPARE statements will re-use the cached statement assuming the environment does not change. (See "Catalog Cache Size (catalogcache_sz)" on page 473 and "Package Cache Size (pckcachesz)" on page 479 for information about a cache that can improve performance when working with dynamic SQL statements.)

- For **static** SQL statements, your testing should compare the statement run times.

  Note: While you may also be interested in the compile time of static SQL, the total (compile and run) time for the statement is difficult to use in any meaningful context. Comparing the total time does not recognize the fact that a static SQL statement can be run many times for each time it is bound and that it is generally not bound during run time.

Quickly Retrieving the First Few Rows Using OPTIMIZE FOR n ROWS

A SELECT statement defines a set of rows which satisfy the search criteria. The DB2 optimizer assumes the application will retrieve all the qualifying rows. This assumption is most appropriate in OLTP and batch environments. However, in "browse" applications it is common for a query to define a very large potential answer set but only retrieve the first few rows, typically only as many rows as are required to fill the screen.

The default assumption made by the optimizer may not be the best for these browse applications. The OPTIMIZE FOR clause provides a mechanism for an application to declare its intent to retrieve only a subset of the result or to give priority to the retrieval of the first few rows. Once this intent is understood, the optimizer can give preference to access plans that minimize the response time for retrieving the first few rows. Also, the number of rows that are sent to the client as a single block (see "Row Blocking" on page 291) are bounded by the value of "n" in the OPTIMIZE FOR clause. Therefore,

the OPTIMIZE FOR clause affects both how the qualifying rows are retrieved from the database by the server, and how the qualifying rows are returned to the client.

For example, suppose you are querying the employee table for the employees with the highest salary on a regular basis.

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
FROM EMPLOYEE
ORDER BY SALARY DESC
```

You have defined a descending index on the SALARY column. However, since employees are ordered by employee number, the salary index is likely to be very poorly clustered. The optimizer, in trying to avoid many random synchronous I/Os, would likely choose to use the list prefetch access method (see “Understanding List Prefetching” on page 407) which requires the row identifiers of all rows that qualify to be sorted. This can cause a delay before the first qualifying rows can be returned to the application. By adding the OPTIMIZE FOR clause to the statement as follows:

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
FROM EMPLOYEE
ORDER BY SALARY DESC
OPTIMIZE FOR 20 ROWS
```

the optimizer would likely choose to use the SALARY index directly with the knowledge that in all likelihood only the twenty employees with the highest salaries would be retrieved. Regardless of how many rows could be blocked, a block of rows is returned to the client every twenty rows.

Use of the OPTIMIZE FOR clause causes the optimizer to favor access plans that avoid bulk operations or operations that interrupt the flow of rows, such as sorts. You are most likely to influence an access path by using OPTIMIZE FOR 1 ROW. As a result, using this clause could have the following effects:

- Join sequences with composite inners are less likely since they require a temporary table.
- The join method could change. A nested loop join is the most likely choice, because it has low overhead cost and is usually more efficient if you only want to retrieve a few rows.
- An index that matches the ORDER BY clause is more likely to be picked. This occurs because no sort would be needed for the ORDER BY.
- List prefetch is less likely to be picked since this access method requires a sort.
- 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.

Although the OPTIMIZE FOR clause applies to all optimization classes (see “Adjusting the Optimization Class” on page 283), it works best for optimization class 3 and higher. The use of the greedy join enumeration method (see “Search Strategies for Selecting Optimal Join” on page 363) in optimization classes below 3 sometimes results in

access plans for multi-table joins that do not lend themselves to quickly retrieving the first few rows.

The OPTIMIZE FOR clause does not prevent you from retrieving all the qualifying rows. However the total elapsed time to retrieve all the qualifying rows may be significantly greater than if the optimizer had been allowed to optimize for the entire answer set.

If you have a packaged application that uses the call level interface (DB2 CLI or ODBC) it is possible to have DB2 CLI automatically append an OPTIMIZE FOR clause to the end of each query statement using the OPTIMIZEFORNROWS keyword in the `db2cli.ini` configuration file. For additional information refer to the *CLI Guide and Reference* manual.

Row Blocking

Row blocking is a technique that reduces database manager overhead by retrieving a *block* of rows in a single operation. These rows are stored in a cache, and each FETCH request in the application gets the next row from the cache. When all the rows in a block have been processed, another block of rows is retrieved by the database manager.

The cache is allocated when an application issues an OPEN CURSOR request and is deallocated when the cursor is closed. The size of the cache is determined by a configuration parameter which is used to allocate memory for the I/O block. The parameter used depends on whether the client is local or remote:

- For *local applications*, the parameter *aslheapsz* is used to allocate the cache for row blocking. (See “Application Support Layer Heap Size (aslheapsz)” on page 492 for information about this parameter.)
- For *remote applications*, the parameter *rqrioblk* on the client workstation is used to allocate the *cache* for row blocking. The cache is allocated on the database client. (See “Client I/O Block Size (rqrioblk)” on page 493 for information about this parameter.)

For *local* applications, you can use the following formula to estimate how many rows are returned per block, where:

- *aslheapsz* is in pages of memory
- 4096 is the number of bytes per page
- *orl* is the output row length in bytes:

```
Rows per block = aslheapsz * 4096 / orl
```

For *remote* applications, you can use the following formula to estimate how many rows are returned per block, where:

- *rqrioblk* is in bytes of memory
- *orl* is the output row length in bytes:

```
Rows per block = rqrioblk / orl
```

Note that if you use the OPTIMIZE FOR *n* ROWS clause in a SELECT statement, the number of rows per block will be the minimum of the following:

- The value calculated in the above formula
- The value of *n* in the OPTIMIZE FOR clause

Use the BLOCKING option on the PREP and BIND commands to specify one of the following types of row blocking:

UNAMBIG	Blocking occurs for read-only cursors and cursors not specified as "FOR UPDATE OF." Ambiguous cursors are treated as updateable.
ALL	Blocking occurs for read-only cursors and cursors not specified as "FOR UPDATE OF." Ambiguous cursors are treated as read-only.
NO	Blocking does not occur for any cursors. Ambiguous cursors are treated as read-only.

For details of these types of row blocking, refer to the PREP and BIND command descriptions in the *Command Reference* manual.

If no option is specified on the PREP and BIND commands, the default row blocking type is UNAMBIG. For the command line processor and call level interface, the default row blocking type is ALL.

Refer to the *SQL Reference* for more information about cursors.

Tuning Queries

This section provides specific considerations and guidelines to help you fine-tune the SQL statements in an application program. As a general rule, these guidelines may help design a program that minimizes the use of system resources and the amount of time needed to access data in a very large table. Depending on the amount of optimization that takes place when the SQL statement is compiled, you may not need to fine-tune your SQL statements. The SQL compiler can rewrite your SQL into more efficient forms. See "Query Rewrite by the SQL Compiler" on page 342 and "Adjusting the Optimization Class" on page 283.

It is also important to note that the access plan chosen by the optimizer is also affected by other factors, including environmental considerations and system catalog statistics. If you conduct benchmark testing of the performance of your applications, you can make adjustments that can improve the access plan.

Using a select-statement

The SQL language is a high-level language with much flexibility. As a result, different *select-statements* can be written to retrieve the same data. However, the performance can vary for the different forms and the different classes of optimization.

It is important to note the SQL compiler (including the query rewrite and optimization phases) must choose an access plan that will produce the result set for the query you have coded. Therefore, as noted in many of the following guidelines, you should code

your query to obtain only the data that you need. This ensures that the SQL compiler can choose the best access plan for your needs.

The guidelines for using a *select-statement* are:

- Specify only those columns that are needed in the select list. Although it may be simpler to specify all columns with an asterisk (*), needless processing and returning of unwanted columns can result.
- Limit the number of rows selected by using predicates to restrict the answer set to only those rows that you require. (See “Predicate Terminology” on page 357 for more information about the different types of predicates and their relative impact on performance.)
- When the number of rows you want to use is significantly less than the total number of rows that could be returned, specify the OPTIMIZE FOR clause for the *select-statement*. This clause affects both the choice of access plans as well as the number of rows that are blocked in the communication buffer. (For more information, see “Row Blocking” on page 291.)
- Specifying the FOR READ ONLY (or FOR FETCH ONLY) clause can improve performance by allowing your query to take advantage of row blocking. It can also improve data concurrency since exclusive locks will never be held on the rows retrieved by a query with this clause specified. It also allows additional query rewrites to take place.
- Specifying the FOR UPDATE OF clause can also improve performance, for cursors that will be updated, by allowing the database manager to initially choose more appropriate locking levels, thus avoiding potential deadlocks (see “Deadlocks” on page 276) and lock conversions (see “Lock Conversion” on page 275).
- Avoid numeric data type conversions whenever possible. When comparing values, it may be more efficient to use items that have the same data type. If conversions are necessary, inaccuracies due to limited precision, and performance costs due to run-time conversions, may result.

  If possible, use the following data types:

  – Character rather than varying character for short columns
  – Integer rather than float or decimal
  – Datetime rather than character.
  – Numeric rather than character.
- SQL statements containing clauses or operations such as DISTINCT, or ORDER BY, require data to be ordered to perform the operation. If you want to decrease the chances that a sort operation will be used, omit the specification of these clauses if they are not required.
- To check for existence of rows in a table, do not use:

  ```
  SELECT COUNT(*) FROM  . . . .
  ```

  and check for a value of nonzero unless you know that the table will be very small. As the table gets larger, counting all the rows will impact performance. Instead it is suggested that you try to select a single row. This can be done by either opening a

cursor and fetching one row, or by doing a single-row (SELECT INTO) selection. (Remember to check for the SQLCODE -811 error if more than one row is found from the *select-statement*.)

- If update activity is low and your tables are large, define indexes on columns that are frequently used as predicates.

The following suggestions apply specifically to *select-statements* that access several tables.

- Use join predicates when joining tables. (A join predicate is a comparison between two columns from different tables in a join.)
- Define indexes on the columns in the join predicate to allow the join to be processed more efficiently. This will also benefit UPDATE and DELETE statements that contain select-statements that access several tables.
- If possible, avoid using expressions or OR clauses with join predicates. In this case, some join techniques cannot be used by the database manager and, as a result, the most efficient join method may not be chosen.
- If possible, ensure that the tables joined are both partitioned on the join column.

For more information see “Join Concepts” on page 359.

Also, refer to the *Embedded SQL Programming Guide* for more information on coding SQL statements with joins and subqueries.

Compound SQL

Compound SQL allows you to group several SQL statements into a single executable block. The SQL statements contained within the block (*sub-statements*) could be executed individually; however, by creating and executing a block of statements, you reduce the database manager overhead. For remote clients, compound SQL also reduces the number of requests that have to be transmitted across the network.

There are two types of compound SQL:

- **Atomic**

  The application receives a response from the database manager when all sub-statements have completed successfully, or when one sub-statement ends in an error. If one sub-statement ends in an error, the entire block is considered to have ended in an error, and any changes made to the database within the block will be rolled back.

- **Not Atomic**

  The application receives a response from the database manager when all sub-statements have completed. All sub-statements within a block are executed regardless of whether or not the preceding sub-statement completed successfully. The group of statements can only be rolled back if the unit of work containing the NOT ATOMIC compound SQL is rolled back.

- Atomic compound SQL is not supported with DB2 Connect
- Compound SQL is supported within Database Application Remote Interface (DARI) routines (stored procedures)
- Compound SQL is supported through:
  - Embedded static SQL (refer to the *SQL Reference* manual)
  - DB2 Call Level Interface (refer to the *CLI Guide and Reference* manual).

Performance Considerations and Character Conversion

When your application and database are not using the same code page, a mapping of the data from one code page to the other code page takes place, if possible. To properly map data between application and database code pages, some data conversion may be required.

This mapping and data conversion introduce a certain amount of overhead into the processing time for applications that are running in a code page that is different from the database code page. Your application's performance can be improved if the application and database are using the same code page or the identity collating sequence.

Character conversion can occur in the following situations:

- When a client or application accessing a database is running in a code page that is different from the code page of the database.
  Database conversion will occur on the database server machine: From the application code page to the database code page; and, from the database code page to the application code page.
- When a client or application importing (or loading) a file runs in a code page different from the file being imported (or loaded).
- When DB2 Connect is used to access data on a DRDA server.

Character conversion will **not** occur for:

- File names.
- Data targeted for, or coming from, a column assigned the FOR BIT DATA attribute, or data used in an SQL operation whose result is FOR BIT or BLOB data.
- Access to a DB2 for OS/2 Version 1.0 or Version 1.2 database server.
- A DB2 product or platform that does not have a supported conversion function to, or from, EUC or UCS-2 installed. You receive an SQLCODE -332 (SQLSTATE 57017) when running your application.

For more information about EUC code page support and National Language Support (NLS) considerations, see the Appendix M, “National Language Support (NLS)” on page 813 appendix later in this book.

Depending on the operating system environment DB2 database managers use a conversion function and conversion tables, or DBCS conversion APIs, when converting multi-byte code pages.

Note: Character string conversions between multi-byte code pages, like DBCS with EUC, may result in either an increase or a decrease in the length of the string.

Code points assigned to different characters in a country's PC DBCS, EUC, and UCS-2 code sets may produce different results when sorting the same characters. If sorting is required across code sets for different countries, you should see the Appendix M, “National Language Support (NLS)” on page 813 appendix later in this book.

Extended UNIX Code (EUC) Code Page Support

Use of host variables that use graphic data in C or C++ applications require special considerations including special precompiler, application performance, and application design issues.

If applications are developed requiring EUC code sets, you should see the *API Reference* manual.

Database and client application support for graphic (that is, double byte character) data must overcome the two bytes wide restriction when dealing with many characters found in both the Japanese and Traditional Chinese EUC code pages. Graphic data from these EUC code pages is stored and manipulated using the UCS-2 code set.

Stored Procedures

In a database application environment, many situations are repetitive; for example, receiving a fixed set of data, performing the same multiple requests against a database, or returning a fixed set of data. Stored procedures permit one call to a remote database to execute a preprogrammed procedure. One call may represent several accesses to the database.

Processing a single SQL statement for a remote database requires sending two transmissions: one request and one receive. However, an application can contain many SQL statements. Without stored procedures, many transmissions are required for an application to complete its work.

When a database client uses a stored procedure, it requires only two transmissions for the entire process, thereby reducing the number of network transmissions. To invoke a stored procedure, the requesting application must connect to the database containing the procedure before calling it.

Typically these stored procedures are run in processes separate from the database agents. This separation requires that the stored procedure and agent processes must communicate through a router. To obtain the best possible performance for a stored procedure, it is possible to identify a stored procedure as being “trusted,” or “not fenced”, and as a result, run the procedure directly in the database agent process. What do we mean by “trusted” and “not fenced”?

- *Not fenced* refers to the fact that there is nothing separating the stored procedure from the database control structures that are used by the database agent.

- *Trusted* indicates that as an administrator, you are confident that the stored procedure will not accidentally or maliciously damage the database control structures. That is, you trust them to operate in a fashion which will not jeopardize your database integrity.

Both of these terms mean the same thing, that is, if your stored procedure is “not fenced”, then your stored procedure is “trusted”. Due to the associated risk of damaging your database, you should only use not fenced stored procedures when you need to obtain the maximum possible performance benefits. In addition, you should ensure that the procedure is well coded and has been thoroughly tested before allowing it to run as a not fenced stored procedure. If a fatal error does occur while running one of these not fenced stored procedures, the database manager will determine whether the error occurred in the application or database manager code, and perform the appropriate recovery.

There are two ways to create a stored procedure as being not fenced:

- Use the CREATE PROCEDURE command and specify the NOT FENCED clause.
- Put the procedure in a special directory, as defined in the *Quick Beginnings* manual for your platform. (This method does not work for Java stored procedures.)

To run a stored procedure, the end-user running the application that calls the procedure must have one of the following privileges at run time:

- EXECUTE or CONTROL privilege for the package associated with the stored procedure
- SYSADM or DBADM authority

For information on writing programs using stored procedures, refer to the *Embedded SQL Programming Guide* manual.

Activating a Database

When a database is started, several types of data are cached. For example, data buffers are cached in the buffer pool, and packages and dynamic SQL statements are cached in the package cache.

If frequent, short periods occur during which no user is connected to the database, and these periods are interspersed with other periods during which a few users are connected to the database, the benefits provided by caching are lost because the cache is frequently destroyed. To avoid this situation, consider activating the database by issuing the following command:

```
db2 ACTIVATE DATABASE database
```

This command activates the specified database and starts up all necessary services, so that the database is available for connection and use by any application. Databases initialized by ACTIVATE DATABASE can be shut down by DEACTIVATE DATABASE or by *db2stop*. For more information about these commands, see the *Command Reference* manual.

Parallel Processing of Applications

“Introduction to Parallelism in DB2 Universal Database” on page xxv describes the various parallel environments supported by DB2. Applications run in some environments (such as those that make use of *Inter-query parallelism*) can effectively ignore the environment. Applications run in other environments, however, can modify certain settings to maximize performance.

A type of parallel environment supported by DB2 is one which requires symmetric multi-processor (SMP) machines. In this environment, more than one processor shares access to the database. This allows parallel execution of complex SQL requests which can be divided among the processors.

You can specify the degree of parallelism to implement when compiling your application by using the CURRENT DEGREE special register, or the DEGREE bind option. "Degree" simply refers to the number of concurrently executing parts of a query. There is no strict relation between the number of processors and the value selected for the degree of parallelism. The total number of processors available for use in your hardware platform need not be requested while running your applications; you can select more or less than this number.

Each degree of parallelism adds to the system memory and CPU overhead.

As a result of using a number of degrees of parallelism, some configuration parameters could be modified to use this parallelism more effectively. Configuration parameters controlling the amount of shared memory and prefetching should be reviewed and modified as necessary in an environment with a high degree of parallelism. See “Parallel” on page 555 for a list of parameters related to parallel operations and partitioned database environments.

There is a database manager configuration parameter, *intra_parallel*, that enables or disables instance parallelism support. The default is "NO" for a uni-processor system and "YES" for SMP machines. An upper limit, or maximum, for the run time degree of parallelism is established in the database configuration parameter, *max_querydegree*. There is a database configuration parameter, *dft_degree*, to specify the default value for the CURRENT DEGREE special register and the DEGREE bind option.

For more information on the application use and implications from using more than one degree of parallelism, see the *Embedded SQL Programming Guide* manual.

Note: The "degree" of parallelism can be set independent of the hardware environment. This means that you can use a degree of parallelism without having an SMP machine. For example, "I/O-bound" queries on a uni-processor machine may benefit from declaring a degree of "2" or more. In this case, the uni-processor may not have to wait for input or output tasks to complete before working on a new query. Declaring a degree of "2" or more does not directly control I/O parallelism on a uni-processor machine. Utilities such as LOAD can control I/O parallelism independent from such a declaration.

In many cases, *database agents* are used to coordinate parallel execution. See “Database Agents” on page 417 for more information, and a list of the various database manager configuration parameters.

Chapter 10. Environmental Considerations

In addition to the factors you should consider when you are designing and coding your application (described in Chapter 9, “Application Considerations” on page 265), there are environmental factors that can influence the access plan chosen for your application:

- Configuration Parameters Affecting Query Optimization
- Nodegroup Impact on Query Optimization
- Table Space Impact on Query Optimization
- Index Management

Also refer to Chapter 11, “System Catalog Statistics” on page 311 for more information about factors that affect the SQL optimizer.

When tuning your applications and environment, you **should** rebind your applications after you make changes in any of the above areas. This ensures that the best access plan is being used.

Configuration Parameters Affecting Query Optimization

Several configuration parameters affect the access plan chosen by the SQL compiler. Many of these are appropriate to a single-partition database and some are only appropriate to a partitioned database. When working with configuration parameters in a partitioned database, it is recommended that the values used for each parameter be the same on all partitions.

Following is a list of configuration parameters that affect the access plan chosen by the SQL compiler:

- “Buffer Pool Size (buffpage)” on page 470.

  When selecting the access plan, the optimizer considers the I/O cost of fetching pages from disk to the buffer pool. In its calculations, the optimizer will estimate the number of I/Os required to satisfy a query. This estimate includes a prediction of buffer pool usage, since additional physical I/Os are not required to read rows in a page that is already in the buffer pool. The optimizer considers the value of the *npages* column in the BUFFERPOOLS system catalog tables in estimating whether a page will be found in the buffer pool.

  The I/O costs of reading the tables can have an impact on :

  – How two tables are joined, as described in “Outer versus Inner Determination” on page 361.
  – Whether an unclustered index will be used to read the data (see “Index Clustering” on page 355).

  You can have more than one buffer pool in a database. You can also have more than one buffer pool in a partitioned database. The new buffer pool can be selectively added to each of the partitions in the database or across all partitions.

The *npages* column in the BUFFERPOOLS and BUFFERPOOLSNODE system catalog tables are used for estimating in a partitioned database.

- "Default Query Optimization Class (dft_queryopt)" on page 543.

  When compiling SQL queries, you can use the query optimization class to direct the optimizer to use different degrees of optimization. For more information on selecting a suitable query optimization class, see "Adjusting the Optimization Class" on page 283.

- "Average Number of Active Applications (avg_appls)" on page 509.

  The *avg_appls* parameter is used by the SQL optimizer to help estimate how much of the buffer pool will be available at run-time for the access plan chosen. Higher values for this parameter can influence the optimizer to choose an access plan for queries that will be more conservative in its buffer pool usage. A value of 1 for this parameter will cause the optimizer to treat the entire buffer pool as being available to the application.

- "Sort Heap Size (sortheap)" on page 482.

  A sort is considered to be "piped" if it does not require a temporary table to store the final, sorted list of data. That is, the results of the sort can be read in a single, sequential access. Piped sorts result in better performance than non-piped sorts and will be used if possible. (See "Influence of Sorting on the Optimizer" on page 372 for a definition of non-piped sorts compared to piped sorts.)

  When choosing an access plan, the optimizer estimates the cost of the sort operations, including evaluating whether a sort can be piped, by:

  – Estimating the amount of data to be sorted
  – Looking at the *sortheap* parameter to determine if there is enough space for the sort to be piped.

- "Maximum Storage for Lock List (locklist)" on page 477 and "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 500.

  When the isolation level (see "Concurrency" on page 265) being used is **repeatable read (RR)**, the SQL optimizer will consider the values of the *locklist* and *maxlocks* parameters to determine whether it is likely that row level locks will be escalated to a table level lock. If the optimizer predicts that lock escalation will occur for a table access, then it will choose a table level lock for the access plan, rather than incurring the overhead of lock escalation during the execution of the query.

- "CPU Speed (cpuspeed)" on page 565.

  The CPU speed is used by the SQL optimizer to estimate the cost of performing certain operations. The optimizer uses these CPU cost estimations along with various I/O cost estimations to select the best access plan for a query.

  The CPU speed of a machine can have a significant influence on the access plan chosen. This configuration parameter is automatically set to an appropriate value when the database is installed or migrated. You should **only** adjust this parameter if you are modelling a production environment on a test system, or to assess the impact of a hardware change. Using this parameter to model a different hardware

environment allows you to observe the access plan that will be chosen for that environment.

- "Statement Heap Size (stmtheap)" on page 484.

  The size of the statement heap does not influence the optimizer in choosing different access paths; however, it can affect the amount of optimization that will be performed for complex SQL statements.

  If the *stmtheap* parameter is not set large enough, you may receive an SQL warning indicating that there is not enough memory available to process the statement. For example, SQLCODE +437 (SQLSTATE 01602) can indicate that the amount of optimization that has been used to compile a statement is less than the amount that you requested when you specified the query optimization class. (See Adjusting the Optimization Class for more information.)

- "Maximum Query Degree of Parallelism (max_querydegree)" on page 559 .

  When this parameter has a value of "ANY", then the optimizer chooses the degree of parallelism to be used. If other than "ANY" is present, then the user-specified value is used to determine the degree of parallelism for the application.

- "Communications Bandwidth (comm_bandwidth)" on page 565 .

  Communications bandwidth is used by the optimizer to determine access paths. The optimizer uses the value in this parameter to estimate the cost of performing certain operations between the database partition servers of a partitioned database.

For additional information, see "Tuning Configuration Parameters" on page 459.

Nodegroup Impact on Query Optimization

In partitioned databases, collocation of tables is recognized by the optimizer and used when determining the best access plan for a query. The assumption is that tables that are frequently involved in join queries should, when divided among partitions in a partitioned database, ideally have the rows from each table being joined located on the same database partition. During the join operation, the collocation of the data from both tables that are part of the join would prevent the need to move data from one partition to another. Placing both tables in the same nodegroup ensures that the data from the tables is collocated together.

See "Table Collocation" on page 37 for more information on collocating tables.

Also, within a partitioned database, the spreading of the data over more partitions reduces the estimated time (or cost) to execute a query. The number of tables, the location of the data in those tables, and the type of query (whether a join is required as noted above) all affect the cost of the query.

Table Space Impact on Query Optimization

Certain characteristics of your table spaces can affect the access plan chosen by the SQL compiler:

- Physical disk characteristics

  Physical disk characteristics can have a significant impact on the I/O cost associated when executing a query. When selecting an access plan the SQL optimizer considers these I/O costs, including any cost differences for accessing data from different table spaces. Two columns in the SYSCAT.TABLESPACES system catalog are used by the optimizer to help estimate the I/O costs of accessing data from a table space:

  – OVERHEAD, which provides an estimate (in milliseconds) of the time required by the physical disk before any data is read into memory. This overhead activity includes the disk's I/O controller overhead as well as the disk latency time, which includes the disk seek time.

    You may use the following formula to help you estimate the overhead cost:

    ```
    OVERHEAD = average seek time in milliseconds
               + 0.5 * rotational latency
    ```

    where:

    - 0.5 represents an average overhead of one half rotation
    - Rotational latency is calculated, in milliseconds for each full rotation, as follows:

      ```
      1 / RPM * 60 * 1000
      ```

      where you:
      - Divide by rotations per minute to get minutes per rotation
      - Multiply by 60 seconds per minute
      - Multiply by 1000 milliseconds per second.

  – TRANSFERRATE, which provides an estimate (in milliseconds) of the time required to read one page of data into memory.

    You may use the following formula to help you estimate the transfer cost in milliseconds per page:

    ```
    TRANSFERRATE = 1 / spec_rate * 1000 / 1,024,000 * 4096
    ```

    where:

    - `spec_rate` represents the disk specification for the transfer rate, in MB per second
    - Divide by `spec_rate` to get Seconds per MB
    - Multiply by 1000 milliseconds per second
    - Divide by 1,024,000 bytes per MB
    - Multiply by 4096 bytes per page

  Each of the containers assigned to a table space may reside on different physical disks. For best results, all physical disks used for a given table space should have the same OVERHEAD and TRANSFERRATE characteristics. If these

characteristics are not the same, you should use the average when setting the values for OVERHEAD and TRANSFERRATE.

You can obtain media specific values for these columns from the hardware specifications or through experimentation. These values may be specified on the CREATE TABLESPACE and ALTER TABLESPACE statements.

This I/O cost information could influence the optimizer in a number of ways, including whether or not to use an index to access the data, and which table to select for the inner and outer tables in a join.

- Prefetching

  When considering the I/O cost of accessing data from a table space, the optimizer will also consider the potential impact that prefetching data and index pages from disk can have on the query performance. Prefetching data and index pages can reduce the overhead and waiting time associated with reading the data into the buffer pool. For more information, see “Prefetching Data into the Buffer Pool” on page 405.

  The optimizer uses the information from the PREFETCHSIZE and EXTENTSIZE columns in SYSCAT.TABLESPACES to estimate the amount of prefetching that will occur for a table space. The EXTENTSIZE can only be set when creating a table space (for example using the CREATE TABLESPACE statement), while PREFETCHSIZE can be set when creating a table space and also using the ALTER TABLESPACE statement.

The following shows an example of the syntax to change the characteristics of the RESOURCE table space:

```
ALTER TABLESPACE RESOURCE
  PREFETCHSIZE 64
  OVERHEAD     19.3
  TRANSFERRATE 0.9
```

After making any changes to your table spaces you should consider rebinding your applications and use the RUNSTATS utility to collect the latest statistics about the indexes to ensure the best access plans are being used.

Index Management

It is important to remember that you do not decide when an index should be used; the database manager makes the decision based on the available table and index information. However, you play an important role in the process by creating the necessary indexes that can improve performance. It is also important for you to collect statistics about the indexes (using the RUNSTATS utility) after you create an index, or change the prefetch quantity (as mentioned above), and on an ongoing basis to keep the statistics up to date. This means you must understand the kinds of indexes that you can create and the ways to create them.

Indexing versus No Indexing

For each table referenced in a database query, if no index exists on the table, then a table scan must be performed on that table. The larger the table, the longer a table scan takes. A *table scan* occurs when the database manager sequentially accesses every row of a table. This can be compared to an *index scan* that occurs when the database manager accesses data using an index. (See “Index Scan Concepts” on page 349.)

An index will be selected for use, if the optimizer estimates that an index scan will be faster than a table scan. Index files generally are smaller and require less time to read than an entire table, particularly as tables grow larger. In addition, the entire index may not need to be scanned. The predicates applied to the index reduce the number of rows to be read from the data pages.

Each index entry consists of a search-key value and a pointer to the row containing that value. The values are arranged in ascending or descending order of the search-key value, which makes it possible to bracket the search, given the right predicates. An index can also be used to obtain rows in an ordered sequence, eliminating the need for the database manager to sort the rows after they are read from the table.

Note: You cannot control whether an index is used by the database manager. For example, the result of a query cannot be guaranteed to be produced in an ordered sequence simply by the existence of an index on the table being queried. The database manager may use this index during the processing of the query but is not required to. Only the existence of an ORDER BY clause can “guarantee” the order of a result set.

Indexes can reduce access time significantly; however, indexes can also have adverse effects on performance. Before creating indexes, consider the effects of multiple indexes on disk space and processing time:

- Each index takes up a certain amount of storage or disk space. The exact amount is dependent on the size of the table and the size and number of columns included in the index.
- Each INSERT or DELETE operation performed on a table requires additional updating of each index on that table. This is also true for each UPDATE operation that changes an index key.
- The LOAD utility rebuilds any existing indexes.
- Each index potentially adds an alternative access path for a query, which the optimizer will consider, and therefore increases the query compilation time.

Indexes should be carefully chosen to address the needs of the application program.

To determine whether an index is used in a specific package you may use the SQL Explain facility, described in Chapter 13, “SQL Explain Facility” on page 377.

Guidelines for Indexing

Which indexes should be created depends on the data and its intended uses. The following guidelines can help you determine which indexes would be most useful:

- Define primary keys and unique keys, wherever they apply, by using the CREATE UNIQUE INDEX statement. (Refer to the *SQL Reference* for more information.) Unique indexes can help the optimizer avoid performing certain operations such as sorts.
- Use indexes to optimize frequent queries to tables with more than a few data pages, as can be determined by the NPAGES column in the SYSCAT.TABLES catalog view:
  - Create an index on any column you will use when joining tables.
  - Create an index on any column from which you will be searching for particular values on a regular basis.
- Avoid creating indexes that are partial keys of other index keys on the columns. For example, if there is an index on columns a, b, and c, then a second index on columns a and b is not generally useful.
- Use indexes on foreign keys to improve performance of delete and update operations on the parent table.
- Use indexes on columns that will frequently be used to sort the data.
- In creating a multiple-column index, if you have more than one choice for the first key column, choose the one most often specified with the "=" predicate or specify the columns with the greatest number of distinct values first.
- Creating indexes, arbitrarily on all columns, not only consumes much disk space, but also causes prepare times to be large. This will be particularly true for complex queries, against which an optimization class with dynamic programming join enumeration is used. (See "Adjusting the Optimization Class" on page 283).
- The following provides a *rule-of-thumb* for the typical number of indexes you will define for a table. This number is based on the primary use of your database:
  - For online transaction processing (OLTP) environments, you should only have one or two indexes
  - For query (read-only) environments, you could have more than five indexes
  - For mixed query/OLTP environments, you could have between two and five indexes.

The following are typical circumstances in which creating an index can improve performance:

- An index can be created on columns that are used in WHERE clauses of the queries and transactions that are most frequently processed.

  The WHERE clause:

  ```
  WHERE WORKDEPT='A01' OR WORKDEPT='E21'
  ```

  will generally benefit from an index on `WORKDEPT`, unless those values occur frequently.

- An index can be created on a column or columns to order the rows in collating sequence. Ordering is required not only in the ORDER BY clause, but also by other features, such as the DISTINCT and GROUP BY clauses.

  The following example uses the DISTINCT clause:

```
SELECT DISTINCT WORKDEPT
  FROM EMPLOYEE
```

The database manager can use an index defined for ascending or descending order on WORKDEPT to eliminate duplicate values. This same index could also be used to group values in the following example with a GROUP BY clause:

```
SELECT WORKDEPT, AVERAGE(SALARY)
  FROM EMPLOYEE
GROUP BY WORKDEPT
```

- An index can be created to name each column that is referenced in a statement. When an index is specified in this way, the resulting index-only access means data can be retrieved more efficiently by avoiding table access.

  For example, assume the following SQL statement is issued:

  ```
  SELECT LASTNAME
    FROM EMPLOYEE
   WHERE WORKDEPT IN ('A00','D11','D21')
  ```

  If an index is defined for the WORKDEPT and LASTNAME columns of the EMPLOYEE table, the statement might be processed more efficiently by scanning the index than by scanning the entire table. Note that since the predicate is on WORKDEPT, this column should be the first column of the index.

Performance Tips for Administering Indexes

The following can help you understand how performance can be impacted by properly using and managing indexes:

1. **Index Creation**

   When creating indexes on large tables, and having an SMP machine, consider setting *intra_parallel* to YES (1) or ANY (-1) to take advantage of parallel performance improvements.

   Multiple processors can be used to scan and sort data. The only time when it is not advantageous to have multiple processors during index creation occurs when the *indexsort* database configuration parameter is NO. (The default for the parameter is YES). The parameter controls whether sorting of index keys is done during index creation.

2. **Index Table Space**

   Indexes may be stored in a different table space from that used to store other table data. This can allow for more efficient use of DASD devices by reducing the movement of read/write heads. You can also create your index table spaces so they will be stored on faster physical devices.

   A table space may also be assigned a separate buffer pool which may protect the index pages from being pushed out of the buffer by the presence of lots of data pages.

   When indexes are not placed in separate table spaces, both data and index pages use the same extent size and prefetch quantity. If you use a different table space for indexes, you have the option of selecting different values for all the

characteristics of a table space. Since indexes are typically smaller than tables and are spread over fewer containers, it is common to find smaller extent sizes such as 8 and 16. For more information see, “Index Page Prefetch” on page 356. Use of faster devices for a table space will be considered by the SQL optimizer, as described in “Table Space Impact on Query Optimization” on page 304. For more information about table spaces, see “Designing and Choosing Table Spaces” on page 38.

3. **Degree of Clustering**

   If your SQL statement requires ordering (for example, ORDER BY, GROUP BY, DISTINCT) and there is an appropriate index to satisfy the ordering, there may be times that the database manager does *not* choose the index. This could happen when:

   - Index clustering is poor (see the CLUSTERRATIO and CLUSTERFACTOR columns of SYSCAT.INDEXES)
   - The table is small enough that it is cheaper to scan the table and sort the answer set in memory
   - There are competing indexes for accessing the table.

   To improve the effectiveness of the index, use the REORG utility to cluster an index on particular columns. Note, however, that in general a table can only be clustered on one index. Your tables and indexes should be built in the sequence of the clustering index for that table.

4. **RUNSTATS Utility**

   After creating a new index, you should use the RUNSTATS utility to collect index statistics. These statistics allow the optimizer to determine whether using the index can improve access performance. See “ Collecting Statistics using the RUNSTATS Utility” on page 312 for more information on this topic.

5. **Reorganizing an Index**

   To get the best performance you can from your indexes, you should consider reorganizing your indexes periodically. Updates to your tables may cause index page prefetch to become less effective. To keep the effectiveness of index page prefetch you must reorganize the index.

   You can reorganize the index by either dropping and re-creating the index, or by using the REORG utility. For more information, see “Reorganizing Table Data” on page 415.

   Dropping and re-creating the index gets a new set of pages that are roughly contiguous and sequential. This improves index page prefetch when it occurs.

   Although more costly to accomplish, the REORG utility also ensures clustering of the data pages. This clustering has greater benefit for index scans accessing a significant number of data pages.

   If you work in a symmetric multi-processor (SMP) system environment, the REORG utility will use multiple processors when *intra_parallel* is YES or ANY.

6. **Use EXPLAIN**

Periodically, run EXPLAIN on your most frequently used queries and check that each of your indexes is used at least once. If an index is not used in any query, consider dropping that index.

Also, use EXPLAIN to see if table scans on large tables are processed as the inner of nested loop joins. This would indicate that an index on the join predicate column is either missing or thought to be ineffective at applying the join predicate. Or, perhaps the join predicate is not present.

Chapter 11. System Catalog Statistics

When optimizing SQL queries, the decisions made by the SQL compiler are heavily influenced by the optimizer's model of the database contents. This data model is used by the optimizer to estimate the costs of alternative access paths that could be used to resolve a particular query.

A key element in the data model is the set of statistics gathered about the data contained in the database and stored in the system catalog tables. This includes statistics for tables, indexes, columns, and user-defined functions (UDFs). A change in the data statistics can result in a change in the choice of access plan selected as the most efficient method of accessing the desired data. After running statistics, you may want to rebind applications.

Examples of the statistics available which help define the data model to the optimizer include:

- The number of pages in a table and the number of pages that are not empty
- The degree to which rows have been moved from their original page to other (overflow) pages.
- The number of rows in a table
- The number of distinct values in a column
- The degree of clustering of an index. That is, the extent to which the physical sequence of rows in a table follows an index.
- The number of index levels and the number of leaf pages in each index
- The number of occurrences of frequently used column values (see “Collecting and Using Distribution Statistics” on page 318)
- The distribution of column values across the range of values present in the column (see “Collecting and Using Distribution Statistics” on page 318)
- Cost estimates for user-defined functions (UDFs).

Statistics for objects are updated in the system catalog tables only when explicitly requested. Some or all of the statistics may be updated by:

- Using the RUNSTATS (run statistics) utility (see “ Collecting Statistics using the RUNSTATS Utility” on page 312)
- Using LOAD, with statistics collection options specified
- Coding SQL UPDATE statements that operate against a set of predefined catalog views (see “User Update-Capable Catalog Statistics” on page 330). Note that statistics for user-defined functions must be updated using this technique (see “Updating Statistics for User-Defined Functions” on page 335). Except for UDFs, the catalogs should only be updated manually for modelling a production environment on a test system or for “what-if analysis.” Statistics should not be updated on production systems.

Additional Information:

The SYSCAT and SYSSTAT catalogs contain information on the statistics gathered. See Appendix I, “Catalog Views” on page 697:

- For information about all the catalog views and the columns they contain.
- For information about all the update-capable catalog views and the columns they contain. You can also refer to this section if you are only interested in the statistical columns of the catalog table.
- For information about table statistics.
- For information about column statistics.
- For information about column distribution statistics.
- For information about index statistics.
- For information about user-defined function statistics.

Collecting Statistics using the RUNSTATS Utility

The RUNSTATS utility updates statistics in the system catalog tables to help with the query optimization process. Without these statistics, the database manager could make a decision that would adversely affect the performance of an SQL statement. The RUNSTATS utility allows you to collect statistics on the data contained in the tables, indexes, or both tables and indexes.

Use the RUNSTATS utility to collect statistics based on both the table and the index data to provide accurate information to the access plan selection process in the following situations:

- When a table has been loaded with data, and the appropriate indexes have been created.
- When a table has been reorganized with the REORG utility.
- When there have been extensive updates, deletions, and insertions that affect a table and its indexes. ("Extensive" in this case may mean that 10 to 20 percent of the table and index data has been affected.)
- Before binding application programs whose performance is critical
- When comparison with previous statistics is desired. Running statistics on a periodic basis permits the discovery of performance problems at an early stage, as described below.
- When the prefetch quantity is changed.
- When you have used the REDISTRIBUTE NODEGROUP utility.

When you are working in a partitioned database, collect the statistics related to a table and its indexes by executing the RUNSTATS operation at a single node. (The node at which the utility executes is determined by whether the node at which you issue the command contains table data or not. See "The Database Partition Where RUNSTATS is Executed" on page 313 for details.) Because the statistics stored in the catalogs are supposed to represent table-level information, the node-level statistics collected by the database manager are multiplied where appropriate by the number of nodes across which the table is partitioned. This provides an approximation of the actual statistics that would be collected by executing RUNSTATS at every node and aggregating these statistics.

Note: The DB2 query optimizer assumes that attribute values (data) are placed equally and evenly across the database partitions of the system. If the placement of data is not equal, you should run this command on a database partition that you think has a representative table distribution.

The Database Partition Where RUNSTATS is Executed

When you invoke RUNSTATS on a table, you must be connected to the database in which the table is stored, but the database partition from which you issue the command does not have to contain a partition for this table:

- If you issue RUNSTATS from a database partition that contains a partition for the table, the utility executes at this database partition.
- If you issue RUNSTATS from a database partition that does not contain a table partition, the request is sent to the first database partition in the nodegroup that holds a partition for the table. The utility then executes at this database partition.

Analyzing Statistics

Analyzing the statistics can indicate when reorganization is necessary. Some of these indications are:

- Clustering of indexes

  If cluster ratio statistics are collected, their value will be in the range from 0 to 100. If cluster factor statistics are collected, their value will be a number between 0 and 1. Only one of these two clustering statistics will be recorded in the SYSCAT.INDEXES catalog. In general, only one of the indexes in a table can have a high degree of clustering. A value of -1 is used to indicate that no statistics are available.

  If you wish to compare ratio values, multiply the cluster factor by 100 to obtain a percentage value for the amount of clustering.

  Index scans that are **not** index-only accesses might perform better with higher cluster ratios. A low cluster ratio leads to more I/O for this type of scan, since after the first access of each data page, it is less likely that the page is still in the buffer pool the next time it is accessed. Increasing the buffer size can improve the performance of an unclustered index. (See “Understanding List Prefetching” on page 407 for information about how the database manager can improve index scan performance for indexes with low cluster ratios and see “Index Clustering” on page 355 for information about how the optimizer uses index statistics.)

  If the table data was initially clustered with respect to a certain index, and the above clustering information indicates that the data is now poorly clustered for that same index, you may wish to reorganize the table to re-cluster the data with respect to that index.

- Overflow of rows

  The overflow number indicates the number of rows that do not fit on their original pages. This can occur when VARCHAR columns are updated with longer values. In such cases, a pointer is kept at the row's original location. This can hurt performance, because the database manager must follow the pointer to find the row's contents, which increases the processing time and may also increase the number of I/Os.

As the number of overflow rows grows higher, the potential benefit of reorganizing your table data also increases. Reorganizing the table data will eliminate the overflowing of rows.

- Comparison of file pages

  The number of pages with rows can be compared with the total number of pages that a table contains. Empty pages will be read for a table scan. Empty pages can occur when entire ranges of rows are deleted.

  As the number of empty pages grows higher, so does the need for a table reorganization. Reorganizing the table can compress the amount of space used by a table, by reclaiming these empty pages. In addition to more efficient use of disk space, reclaiming unused pages can also improve the performance of table scan, since fewer pages will be read into the buffer pool.

- Number of leaf pages

  The number of leaf pages predicts how many index page I/Os are needed for a complete scan of an index.

  Random update activity can cause page splits to occur that increase the size of the index beyond the minimum amount of space required. When indexes are rebuilt during the reorganization of a table, it is possible to build each index with the minimum amount of space possible. For more information on the minimum space requirements for an index, see “Creating an Index” on page 95 and “Index Management” on page 305.

  Note: A default of ten percent free space is left on each index page when the indexes are rebuilt. The environment variable `DB2_INDEX_FREE` can be used to establish a value other than the default for the amount of free space for each index page. The maximum amount of free space for each index page is sixty percent.

RUNSTATS can also help you determine how performance is related to changes in your database. The statistics show the data distribution within a table. When used routinely, RUNSTATS provides data about tables and indexes over a period of time, thereby allowing performance trends to be identified for your data model as it evolves over time.

Ideally, you should rebind application programs after running statistics, because the query optimizer may choose a different access plan given the new statistics.

If you do not have enough time available to collect all of the statistics at one time, you may choose to periodically run RUNSTATS to update only a portion of the statistics that could be gathered. If inconsistencies are found as a result of activity on the table between the periods where you run RUNSTATS with a selective partial update, then a warning message (SQL0437W, reason code 6) is issued. For example, you first use RUNSTATS to gather table distribution statistics. Subsequently, you use RUNSTATS to gather index statistics. If inconsistencies are detected as a result of activity on the table, then the table distribution statistics are dropped and the warning message is issued. It is recommended that you run RUNSTATS to gather table distribution statistics when this happens.

You should periodically use RUNSTATS to gather both table and index statistics at once, to ensure that the index statistics are synchronized with the table statistics. Index statistics retain most of the table and column statistics collected from the last run of RUNSTATS. If the table has been modified extensively since the last time its table statistics were gathered, gathering only the index statistics for that table will leave the two sets of statistics out of synchronization.

You may wish to collect statistics based only on index data in the following situations:

- A new index has been created since the utility was performed and you do not want to re-collect statistics on the table data.
- There have been a lot of changes to the data that affect the first column of an index.

The RUNSTATS utility allows you to collect varying levels of statistics. For tables, you can collect basic level statistics or you can also collect distribution statistics for the column values within a table (see “Collecting and Using Distribution Statistics” on page 318). For indexes, you can collect basic level statistics or you can also collect detailed statistics which can help the optimizer better estimate the I/O cost of an index scan. (See “Index Clustering” on page 355 for information about these “detailed” statistics).

Note: Statistics are not collected for LONG or large object (LOB) columns.

The following tables show the catalog statistics that are updated by the RUNSTATS utility:

Table 29. Table Statistics (SYSCAT.TABLES and SYSSTAT.TABLES)

Statistic	**Description**	**RUNSTATS Option**	
		Table	**Indexes**
FPAGES	number of pages being used by a table	Yes	Yes
NPAGES	number of pages containing rows	Yes	Yes
OVERFLOW	number of rows that overflow	Yes	No
CARD	number of rows in table (cardinality)	Yes	Yes (Note 2)

Note:

1. For a partitioned database, the values for each statistic are estimated from the value of the count at the database partition multiplied by the number of database partitions.
2. If the table has no indices defined and you request statistics for indexes, no new CARD statistics are updated. The previous CARD statistics are retained.

Table 30. Column Statistics (SYSCAT.COLUMNS and SYSSTAT.COLUMNS)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
COLCARD	column cardinality	Yes (Note 1)	Yes (Note 2)
AVGCOLLEN	average length of column	Yes	Yes (Note 2)
HIGH2KEY	second highest value in column	Yes	Yes (Note 2)
LOW2KEY	second lowest value in column	Yes	Yes (Note 2)

Note:

1. COLCARD is estimated for all columns in the table. In a partitioned database, if the column is the single-column partitioning key for the table, the value of the count is estimated as the count at the database partition multiplied by the number of database partitions.
2. Column statistics are gathered for the first column in the index key.

Table 31 (Page 1 of 2). Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
NLEAF	number of index leaf pages	No	Yes (Note 3)
NLEVELS	number of index levels	No	Yes
CLUSTERRATIO	degree of clustering of table data	No	Yes (Note 2)
CLUSTERFACTOR	finer degree of clustering	No	Detailed (Notes 1,2)
DENSITY	Ratio (percentage) of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index (Note 4)	No	Yes
FIRSTKEYCARD	number of distinct values in first column of the index	No	Yes (Note 3)
FIRST2KEYCARD	number of distinct values in first two columns of the index	No	Yes (Note 3)
FIRST3KEYCARD	number of distinct values in first three columns of the index	No	Yes (Note 3)
FIRST4KEYCARD	number of distinct values in first four columns of the index	No	Yes (Note 3)

Table 31 (Page 2 of 2). Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
FULLKEYCARD	number of distinct values in all columns of the index	No	Yes (Note 3)
PAGE_FETCH_PAIRS	page fetch estimates for different buffer sizes	No	Detailed (Notes 1,2)
SEQUENTIAL_PAGES	number of leaf pages located on disk in index key order, with few or no large gaps between them	No	Yes

Note:

1. Detailed index statistics are gathered by specifying the DETAILED clause on the RUNSTATS command, or by specifying `A`, `Y` or `X` for the `statsopt` parameter when calling the RUNSTATS API.
2. CLUSTER_FACTOR and PAGE_FETCH_PAIRS are not collected with the DETAILED clause unless the table is of a respectable size. If the table is greater than about 25 pages, then CLUSTERFACTOR and PAGE_FETCH_PAIRS statistics are collected. In this case, CLUSTERRATIO is -1 (not collected). If the table is a relatively small table, only CLUSTERRATIO is filled in by RUNSTATS while CLUSTERFACTOR and PAGE_FETCH_PAIRS are not. If the DETAILED clause is not specified, only the CLUSTERRATIO statistic is collected.
3. For a partitioned database, the value is estimated from the value of the count at the database partition multiplied by the number of database partitions.
4. This statistic measures the percentage of pages in the file containing the index that belongs to that table. For a table having only one index defined on it, DENSITY should normally be 100. DENSITY is used by the optimizer to estimate how many irrelevant pages from other indexes might be read, on average, if the index pages were prefetched.

Table 32 (Page 1 of 2). Column Distribution Statistics (SYSCAT.COLDIST and SYSSTAT.COLDIST)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
DISTCOUNT	If TYPE is Q, the number of distinct values that are less than or equal to COLVALUE statistics	Distribution (Note 2)	No
TYPE	Indicator of whether row provides frequent-value or quantile statistics	Distribution	No
SEQNO	Frequency ranking of a sequence number to help uniquely identify the row in the table	Distribution	No

Table 32 (Page 2 of 2). Column Distribution Statistics (SYSCAT.COLDIST and SYSSTAT.COLDIST)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
COLVALUE	Data value for which frequency or quantile statistic is collected	Distribution	No
VALCOUNT	Frequency with which the data value occurs in column, or for quantiles, the number of values less than or equal to the data value (COLVALUE)	Distribution	No

Note:

1. Column distribution statistics are gathered by specifying the WITH DISTRIBUTION clause on the RUNSTATS command, or by specifying `A`, `D` or `Y` for the `statsopt` parameter when calling the RUNSTATS API. Note that distribution statistics may **not** be gathered unless there is a sufficient lack of uniformity in the column values.
2. DISTCOUNT is collected only for columns that are the first key column in an index.
3. In a partitioned database, VALCOUNT is the estimated value of the count at the database partition multiplied by the number of database partitions. The exception to this is where the TYPE is 'F' and the column is the single-column partitioning key of the table, in which case VALCOUNT is simply the count at the database partition.

For more information about column distribution statistics, see “Collecting and Using Distribution Statistics.”

Statistics for user-defined functions are not collected by the RUNSTATS utility. You must manually update the statistics for these functions. See “User Update-Capable Catalog Statistics” on page 330 and “Updating Statistics for User-Defined Functions” on page 335.

Collecting and Using Distribution Statistics

The database manager can collect, maintain, and use “frequent-value statistics” and “quantiles,” two types of statistics that estimate, in a concise way, the distribution of the data values in a column. Use of these statistics by the optimizer can lead to significantly more accurate estimates of the number of rows in a column that satisfy given equality or range predicates. These more accurate estimates in turn increase the likelihood that the optimizer will choose an optimal plan.

You may collect statistics about the distribution of these data values by using the WITH DISTRIBUTION clause on the RUNSTATS command. While collecting these additional statistics results in additional overhead for the RUNSTATS utility, the SQL compiler can use this information to help ensure the best access plan is chosen.

In some cases, the database manager will not collect distribution statistics and no error will be returned. For example:

- The *num_freqvalues* and *num_quantiles* configuration parameters are set to zero (0) to indicate that you do not want to collect distribution statistics. For more information about these parameters, see:
  - "How Many Statistics Should You Keep?" on page 322
  - "Number of Frequent Values Retained (num_freqvalues)" on page 544
  - "Number of Quantiles for Columns (num_quantiles)" on page 544.
- The distribution of the data is known without the use of distribution statistics. For example, a column that does not have any data value appearing more than once, that is, each data value in the column is unique.
- The data type is one for which statistics are not collected. That is, the column is defined using a long field or large object data type.
- In the case of quantiles, there is only one non-NULL value in the column.

Distribution statistics are exact for the first column of indexes. For each additional column, the database manager uses hashing and sampling techniques to estimate the distribution statistics because calculating exact statistics would require too much time and memory to be practical. These techniques are accepted statistical methods with accepted degrees of accuracy.

The following topics provide information to help you understand and use these distribution statistics:

- Understanding Distribution Statistics
- When Should You Use Distribution Statistics?
- How Many Statistics Should You Keep?
- How Does the Optimizer Use Distribution Statistics?
- Modelling Production Databases
- Rules for Updating Distribution Statistics for Columns

Understanding Distribution Statistics

For a fixed number N>=1, the *N most frequent values* in a column consist of the data value having the highest frequency (that is, number of duplicates), the data value having the second highest frequency, and so forth, down to the data value having the Nth highest frequency. The corresponding *frequent-value statistics* consist of these "N" data values, together with the frequencies of these values in the column.

The *K-quantile* for a column is the smallest data value, V, such that at least "K" rows have data values less than or equal to V. A K-quantile can be computed by sorting the rows in the column according to increasing data values; the K-quantile is the data value in the Kth row of the sorted column.

For example, consider the following column of data:

```
C1
--
 B
 E
 Y
 B
 F
 G
 E
 A
 J
 K
 E
 L
```

This column can be sorted to obtain the following ordered values:

```
C1'
--
 A
 B
 B
 E
 E
 E
 F
 G
 J
 K
 L
 Y
```

There are nine distinct data values in column C1. For N = 2, the frequent value statistics are:

```
SEQNO    COLVALUE     VALCOUNT
-----    ---------    --------
  1          E            3
  2          B            2
```

If the number of quantiles being collected is 5 (see “Number of Quantiles for Columns (num_quantiles)” on page 544), then the K-quantiles for this column for K = 1, 3, 6, 9, and 12 are:

```
SEQNO    COLVALUE     VALCOUNT
-----    ---------    --------
  1          A            1
  2          B            3
  3          E            6
  4          J            9
  5          Y           12
```

In this example, the 6-quantile is equal to E since the sixth row in the sorted column has a data value equal to E (and 6 rows in the original column have data values less than or equal to E).

The same quantile value may occur more than once, if it is a common value. A maximum of two quantiles will be stored for a given value. The first of these two quantiles has a COLCOUNT that gives the number of rows strictly less than COLVALUE, and the second of the two quantiles gives the number of rows less than or equal to COLVALUE.

When Should You Use Distribution Statistics?

To decide whether distribution statistics should be kept for a given table, two factors should be considered:

1. The use of static or dynamic SQL.

   Distribution statistics are most useful for dynamic SQL and static SQL that does not use host variables. When using SQL with host variables, the optimizer makes limited use of distribution statistics.

2. The lack of uniformity in the data distributions.

   Keeping distribution statistics is advisable if at least one column in the table has a highly “non-uniform” distribution of data and the column appears frequently in equality or range predicates; that is, in clauses such as the following:

   ```
   WHERE C1 = KEY;
   WHERE C1 IN (KEY1, KEY2, KEY3);
   WHERE (C1 = KEY1) OR (C1 = KEY2) OR (C1 = KEY3);
   WHERE C1 <= KEY;
   WHERE C1 BETWEEN KEY1 AND KEY2;
   ```

   There can be two types of non-uniformity in a data distribution, possibly occurring together:

   - One type of non-uniformity occurs when the data, instead of being evenly spread out between the highest and lowest data value, is clustered in some sub-interval, as in the following column, where the data is clustered in the range (5,10):

     ```
          C1
     -------
        0.0
        5.1
        6.3
        7.1
        8.2
        8.4
        8.5
        9.1
       93.6
      100.0
     ```

     It can be useful to keep quantiles when this type of non-uniformity is present.

The following example shows a query that can be used to help determine whether a high degree of non-uniformity exists in a column.

```
SELECT C1, COUNT(*) AS OCCURRENCES
  FROM T1
GROUP BY C1
ORDER BY OCCURRENCES DESC;
```

- Another type of non-uniformity occurs when certain data values have a much higher frequency than other data values, as in a column having data values with the following frequencies:

```
Data Value   Frequency
----------   ---------
    20           5
    30          10
    40          10
    50          25
    60          25
    70          20
    80           5
```

  It can be useful to keep both quantiles and frequent-value statistics when this type of non-uniformity is present.

You may collect distribution statistics by using the WITH DISTRIBUTION clause on the RUNSTATS command, or by specifying `D`, `E`, or `A` for the `statsopt` parameter when calling the RUNSTATS API. For more information, refer to the *Command Reference* or the *API Reference* manuals.

How Many Statistics Should You Keep?

Keeping a large number of column distribution statistics can lead to improved selection of access plans by the optimizer, but the cost of collecting these statistics and compiling your queries increases accordingly. The size of the statistics heap (see “Statistics Heap Size (stat_heap_sz)” on page 485) may place limitations on the number of statistics that can be computed and stored.

When distribution statistics are requested, the database manager stores a default of the 10 most frequent values for a column. Keeping between 10 and 100 frequent values should suffice for most practical situations. Ideally, enough frequent-value statistics should be retained so that the frequencies of the remaining values are either approximately equal to each other or negligible compared to the frequencies of the most frequent values.

To set the number of frequent values to collect, use the *num_freqvalues* configuration parameter, as described in “Number of Frequent Values Retained (num_freqvalues)” on page 544. The database manager may collect less than this number of frequent value statistics, because these statistics will only be collected for data values that occur more than once. If collecting only quantile statistics, this parameter can be set to zero.

When distribution statistics are requested, the database manager stores a default of 20 quantiles for a column. This value guarantees a maximum estimation error of

approximately 2.5% for any simple single-sided range predicate (>, >=, <, or <=), and a maximum error of 5% for any BETWEEN predicate. A rough rule of thumb for determining the number of quantiles is:

- Determine the maximum error that is tolerable in estimating the number of rows of any range query, as a percentage, P
- The number of quantiles should be approximately 100/P if the predicate is a BETWEEN predicate, and 50/P if the predicate is any other type of range predicate (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and of 2% for ">" predicates. In general, at least 10 quantiles should be kept, and more than 50 quantiles should be necessary only for extremely non-uniform data.

To set the number of quantiles, use the *num_quantiles* configuration parameter as described in "Number of Quantiles for Columns (num_quantiles)" on page 544. If collecting only frequent value statistics, this parameter can be set to zero. Setting this parameter to "1" will also result in no quantile statistics being gathered since the entire range of values will fit in one quantile.

How Does the Optimizer Use Distribution Statistics?

Why collect and store distribution statistics? The answer lies in the fact that an optimizer needs to estimate the number of rows in a column that satisfy an equality or range predicate in order to select the least expensive access plan. The more accurate the estimate, the greater the likelihood that the optimizer will choose the optimal access plan. For example, consider the query

```
SELECT C1, C2
  FROM TABLE1
 WHERE C1 = 'NEW YORK'
   AND C2 <= 10
```

and suppose that there is an index on C1 and an index on C2. One possible access plan is to use the index on C1 to retrieve all rows with `C1 = 'NEW YORK'` and then check each retrieved row to see if `C2 <= 10`. An alternative plan is to use the index on C2 to retrieve all rows with `C2 <= 10` and then check each retrieved row to see if `C1 = 'NEW YORK'`. Typically, the primary cost in executing the above query is the cost of the retrieving the rows, and so it is desirable to choose the plan the that requires the minimum number of retrievals. To choose the best plan, it is necessary to estimate the number of rows that satisfy each predicate.

When you do not request distribution statistics, the optimizer maintains only the second-highest data value (HIGH2KEY), second-lowest data value (LOW2KEY), number of distinct values (COLCARD), and number of rows (CARD) for a column. The number of rows that satisfy an equality or range predicate is then estimated under the assumption that the frequencies of the data values in a column are all equal and the data values are evenly spread out over the interval (LOW2KEY, HIGH2KEY). Specifically, the number of rows satisfying an equality predicate `C1 = KEY` is estimated

as CARD/COLCARD, and the number of rows satisfying a range predicate `C1 BETWEEN KEY1 AND KEY2` is estimated as:

```
      KEY2 - KEY1
------------------  x CARD       (1)
 HIGH2KEY - LOW2KEY
```

These estimates are accurate only when the true distribution of data values in a column is reasonably uniform. When distribution statistics are unavailable and either the frequencies of the data values differ widely from each other or the data values are clustered in a few sub-intervals of the interval (LOW_KEY,HIGH_KEY), the estimates can be off by orders of magnitude and the optimizer may choose a less than optimal access plan.

When distribution statistics are available, the errors described above can be greatly reduced by using frequent-value statistics to compute the number of rows that satisfy an equality predicate and using frequent-value statistics and quantiles to compute the number of rows that satisfy a range predicate.

Example of Impact on Equality Predicates:

Consider first a predicate of the form `C1 = KEY`. If `KEY` is one of the N most frequent values, then the optimizer simply uses the frequency of `KEY` that is stored in the catalog. If `KEY` is not one of the N most frequent values, the optimizer estimates the number of rows that satisfy the predicate under the assumption that the (COLCARD - N) non-frequent values have a uniform distribution. That is, the number of rows is estimated as:

```
CARD - NUM_FREQ_ROWS
--------------------          (2)
   COLCARD - N
```

where NUM_FREQ_ROWS is the total number of rows with a value equal to one of the N most frequent values.

For example, consider a column (`C`) for which the frequency of the data values is as follows:

Data Value	Frequency
1	2
2	3
3	40
4	4
5	1

Suppose that frequent-value statistics based on only the most frequent value (that is, N = 1) are available. For this column, CARD = 50 and COLCARD = 5. For the predicate `C = 3`, exactly 40 rows satisfy it. Assuming a uniform data distribution, the number of rows that satisfy the predicate is estimated as 50/5 = 10, an error of -75%. Using frequent-value statistics, the number of rows is estimated as 40, with no error.

Similarly, 2 rows satisfy the predicate `C = 1`. Without frequent-value statistics, the number of rows that satisfy the predicate is estimated as 10, an error of 400%. You may use the following formula to calculate the estimation error (as a percentage):

```
estimated rows  - actual rows
----------------------------  X 100
          actual rows
```

Using the frequent value statistics (N = 1), the optimizer will estimate the number of rows containing this value using the formula (2) given above, for example:

```
(50 - 40)
--------- = 3
 (5 - 1)
```

and the error is reduced by an order of magnitude as shown below:

```
 3 - 2
------- = 50%
   2
```

The number of rows that satisfy a range predicate can be estimated using quantiles, as illustrated by the following examples. Consider a column (`C`) given by:

```
    C
-------
   0.0
   5.1
   6.3
   7.1
   8.2
   8.4
   8.5
   9.1
  93.6
 100.0
```

and suppose that K-quantiles are available for K = 1, 4, 7, and 10:

```
 K   K-quantile
---  ----------
 1      0.0
 4      7.1
 7      8.5
10    100.0
```

First consider the predicate `C <= 8.5`. For the data given above, exactly 7 rows satisfy this predicate. Assuming a uniform data distribution and using formula (1) from above, with KEY1 replaced by LOW2KEY, the number of rows that satisfy the predicate is estimated as:

```
 8.5 - 5.1
---------- x 10 *= 0
93.6 - 5.1
```

where *= means “approximately equal to.” The error in this estimation is approximately -100%.

Using quantiles, the number of rows that satisfy this same predicate (C <= 8.5) is estimated by locating 8.5 as one of the K-quantile values and using the corresponding value of K, namely 7, as the estimate. In this case, the error is reduced to 0.

Now consider the predicate C <= 10. Exactly 8 rows satisfy this predicate. Unlike the previous example, the value 10 is not one of the stored K-quantiles. Assuming a uniform data distribution and using formula (1), the number of rows that satisfy the predicate is estimated as 1, an error of -86%.

Using quantiles, the optimizer estimates the number of rows that satisfy the predicate as r_1 + r_2, where r_1 is the number of rows satisfying the predicate C <= 8.5 and r_2 is the number of rows satisfying the predicate C > 8.5 AND C <= 10.. As in the above example, r_1 = 7. To estimate r_2 the optimizer uses linear interpolation:

```
        100.0 - 10.0
r_2 *= ------------ x (# rows with value > 8.5 and <= 100.0)
        100.0 - 8.5
        100.0 - 10.0
     = ------------ x (10 - 7)
        100.0 - 8.5
    *= 3
```

The final estimate is r_1 + r_2 *= 10, and the absolute error is reduced by more than a factor of 3.

The reason that the use of quantiles improves the accuracy of the estimates in the above examples is that the real data values are "clustered" in the range 5 - 10, but the standard estimation formulas assume that the data values are spread out evenly between 0 and 100.

The use of quantiles also improves accuracy when there are significant differences in the frequencies of different data values. Consider a column having data values with the following frequencies:

Data Value	Frequency
20	5
30	5
40	15
50	50
60	15
70	5
80	5

Suppose that K-quantiles are available for K = 5, 25, 75, 95, and 100:

```
 K     K-quantile
----   ----------
  5       20
 25       40
 75       50
 95       70
100       80
```

Also suppose that frequent value statistics are available based on the 3 most frequent values.

Consider the predicate `C BETWEEN 20 AND 30`. From the distribution of the data values, you can see that exactly 10 rows satisfy this predicate. Assuming a uniform data distribution and using formula (1), the number of rows that satisfy the predicate is estimated as:

```
30 - 20
-------  x 100 = 25
70 - 30
```

which has an error of 150%.

Using frequent-value statistics and quantiles, the number of rows that satisfy the predicate is estimated as `r_1 + r_2`, where `r_1` is the number of rows that satisfy the predicate `(C = 20)` and `r_2` is the number of rows that satisfy the predicate `C > 20 AND C <= 30`. Using formula (2), `r_1` is estimated as:

```
100 - 80
-------- = 5
  7 - 3
```

Using linear interpolation, `r_2` is estimated as:

```
  30 - 20
  ------- x (# rows with value > 20 and <= 40)
  40 - 20
  30 - 20
= ------- x (25 - 5)
  40 - 20
= 10,
```

yielding a final estimate of 15 and reducing the error by a factor of 3.

Collecting and Using Detailed Index Statistics

As an option, you may collect more detailed statistics on indexes that help the optimizer better estimate the cost of accessing a table using that index. This can be done in one of two ways: First, you can use the DETAILED clause on the RUNSTATS command; or, second, you can specify A, Y, or X for the `satsopt` parameter when calling the RUNSTATS API. The DETAILED statistics PAGE_FETCH_PAIRS and CLUSTER_FACTOR will be collected only if the table is of a sufficient size: around 25 pages. In this case, CLUSTER_FACTOR will be a value between 0 and 100; and

CLUSTER_RATIO will be -1 (not collected). For tables smaller than 25 pages, CLUSTER_FACTOR will be a value between 0 and 100; and CLUSTER_RATIO will be -1 (not collected), even if the DETAILED clause is specified for an index on that table.

Understanding Detailed Index Statistics

The DETAILED statistics attempt to capture, in a concise way, the number of physical I/Os that will be required to access the data pages of a table when a complete index scan is performed under different buffer sizes. As RUNSTATS scans through the pages of the index, it models the different buffer sizes, and gathers estimates of how often a page fault occurs. For example, with only 1 (one) buffer page available, every new page reference by the index will result in a page fault, and, in a worse case, every row could reference a different page, resulting in at most CARDINALITY I/Os. At the other extreme, when the buffer is big enough to hold the entire table (subject to the maximum buffer size), then each of the table's NPAGES pages will be physically read exactly once. The number of physical I/Os must therefore be a monotone, non-increasing function of the buffer size.

RUNSTATS fits a piece-wise linear curve to these estimates, which is stored as a string of 11 pairs in the PAGE_FETCH_PAIRS statistic. The first value in each pair is a hypothetical buffer size, and the second value in each pair is the estimated number of physical I/Os to fetch the data pages in a complete scan of the index, with a buffer of that size totally available to that index scan. The optimizer then uses the PAGE_FETCH_PAIRS statistic to estimate the number of physical I/Os for data-page fetches in any complete or partial index scan using that index.

The shape of the curve stored in PAGE_FETCH_PAIRS for an index will depend upon the clustering behavior of that index.

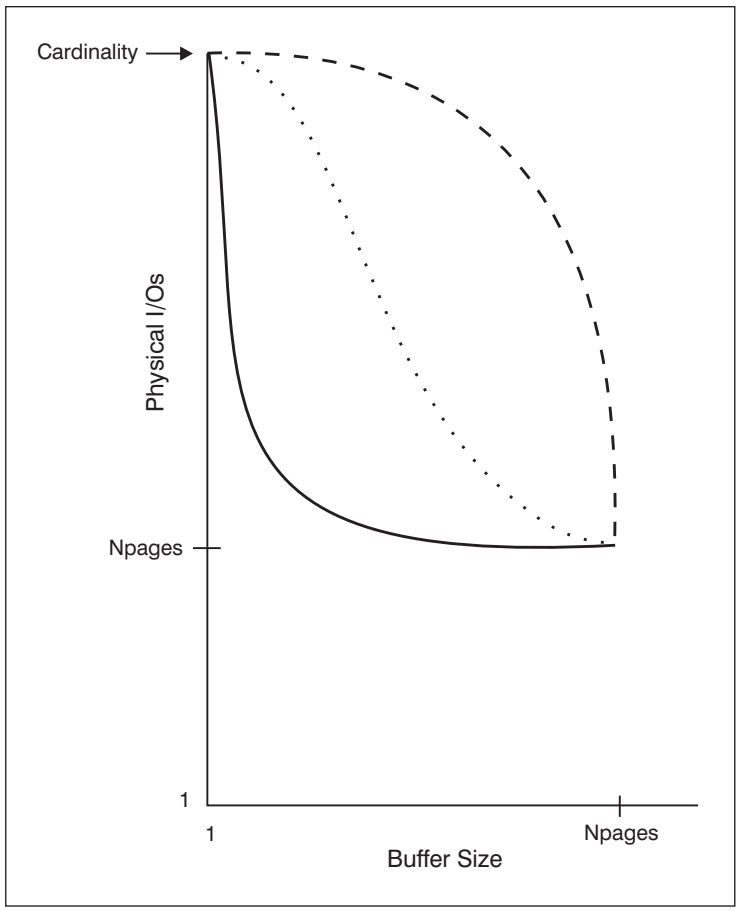


Figure 38. Three Curves for Clustered and Unclustered Indexes

There are three types of curves that are possible:

1. Curve 1 (dashed-line) is a highly-unclustered index that needs a buffer almost as large as the table before re-referenced pages are found in the buffer. This represents a situation in which references to the same page are widely spread throughout the index's key values, so a medium-sized buffer isn't sufficient to avoid re-referencing the same page multiple times. This is the worst scenario, as it requires the most buffer space to perform well. The optimizer is likely to use the list prefetch access strategy for such indexes, in an attempt to cluster the data-page accesses for the qualifying key values of the index. If this index is used frequently, it should be a prime candidate for reorganization.
2. Curve 2 (solid-line) is more locally unclustered. For very small buffers, it is as unclustered as curve 1, but once a few buffer pages are available to contain the most recently referenced data, the data-page hit ratio improves significantly. This represents the somewhat favorable situation in which, although the index isn't

particularly clustered, references to the same data pages are in a close proximity to one another among the index's key values.

3. Curve three (dotted-line) is somewhere between these two extremes, improving at a uniform rate as the buffer is increased. This is usually the more common case for unclustered indexes, and represents what the optimizer will assume in the absence of DETAILED indexes.

When Should You Use Detailed Index Statistics?

You should use DETAILED index statistics when your queries reference columns that are not all in the index. In addition, DETAILED index statistics should be used when:

- There are multiple unclustered indexes with varying degrees of clustering
- The degree of clustering is non-uniform among the key values
- The values in the index are updated non-uniformly.

It may be quite hard to determine these situations without previous knowledge, and without attempting to force an index scan under varying buffer sizes and using the monitor to observe the physical I/Os that result. Probably the cheapest way to determine whether any of these situations are occurring is to collect the DETAILED statistics for an index and retain them if the PAGE_FETCH_PAIRS that result are non-linear.

User Update-Capable Catalog Statistics

The ability to update selected system catalog statistics allows you to:

- Model query performance on a development system using production system statistics
- Perform “what if” query performance analysis.

You should **not** update statistics on a production system because you may hinder the optimizer from finding the best access plan for your query.

To update the values of these statistical columns, use the SQL UPDATE statement against the views defined in the SYSSTAT schema. You can update statistics for:

- Tables for which you hold explicit CONTROL privilege. You can also update statistics for columns and indexes for these tables.
- User-defined functions (UDFs) that you own (see “Updating Statistics for User-Defined Functions” on page 335 for guidance).

You can also update these statistics if your user ID has explicit DBADM authority for the database; that is, your user ID is recorded as having DBADM authority in the SYSCAT.DBAUTH table. Belonging to a DBADM group does not explicitly provide this authority.

Using these views, a DBADM can see statistics rows for all users. A user without DBADM authority can only see those rows which contain statistics for objects over which they have CONTROL privilege.

The following shows an example of updating the table statistics for the EMPLOYEE table:

```
UPDATE SYSSTAT.TABLES
SET  CARD   = 10000,
     NPAGES = 1000,
     FPAGES = 1000,
     OVERFLOW = 2
WHERE TABSCHEMA = 'userid'
  AND TABNAME   = 'EMPLOYEE'
```

You must be careful when updating catalog statistics. Arbitrary updates can have a serious impact on the performance of subsequent queries. You may wish to use any of the following methods to replace any updates you applied to these tables:

- ROLLBACK the unit of work in which the changes have been made (assuming the unit of work has not been committed).
- Using the RUNSTATS utility you can recalculate and refresh the catalog statistics.
- Update the catalog statistics to indicate that statistics have not been gathered. (For example, setting column NPAGES to -1 indicates that the number-of-pages statistic has not been collected.)
- Replace the catalog statistics with the data they contained prior to your update. This method would only be possible if you used the `db2look` tool, as described in “Modelling Production Databases” on page 337, to capture the statistics before you made any changes.

In a some cases, the optimizer may determine that some particular statistical value or combination of values are not valid, it will use default values and issue a warning. Such circumstances are rare, however, since most of the validation is done when updating the statistics.

Additional Information: For information about updating catalog statistics, see:

- “Rules for Updating Catalog Statistics”
- “Rules for Updating Table Statistics” on page 332
- “Rules for Updating Column Statistics” on page 332
- “Rules for Updating Distribution Statistics for Columns” on page 333
- “Rules for Updating Index Statistics” on page 334
- “Updating Statistics for User-Defined Functions” on page 335
- “Modelling Production Databases” on page 337

Rules for Updating Catalog Statistics

When you update catalog statistics, the most important general rule is to ensure that valid values, ranges, and formats of the various statistics are stored in the statistic views. It is also important to preserve the consistency of relationships between various statistics.

For example, COLCARD in SYSSTAT.COLUMNS must be less than CARD in SYSSTAT.TABLES (the number of distinct values in a column can't be greater than the number of rows). Assume that you want to reduce COLCARD from 100 to 25, and CARD from 200 to 50. If you update SYSCAT.TABLES first, you should get an error

(since CARD would be less than COLCARD). The correct order is to update COLCARD in SYSCAT.COLUMNS first, then update CARD in SYSSTAT.TABLES. The situation occurs in reverse if you want to increase COLCARD to 250 from 100, and CARD to 300 from 200. In this case, you must update CARD first, then COLCARD.

When a conflict is detected between an updated statistic and another statistic, an error is issued. However, errors may not always be issued when conflicts arise. In some situations, the conflict is difficult to detect and report in an error, especially if the two related statistics are in different catalogs. For this reason, you should be careful to avoid causing such conflicts.

The most common checks you should make, before updating a catalog statistic, are:

1. Numeric statistics must be -1 or greater than or equal to zero.
2. Numeric statistics representing percentages (for example, CLUSTERRATIO in SYSSTAT.INDEXES) must be between 0 and 100.

Rules for Updating Table Statistics

There are only four statistic values that you can update in sysstat.tables: CARD, FPAGES, NPAGES, and OVERFLOW. Keep in mind that:

1. CARD must be greater than all COLCARD values in SYSSTAT.COLUMNS that correspond to that table.
2. CARD must be greater than NPAGES.
3. FPAGES must be greater than NPAGES.
4. NPAGES must be less than or equal to any "Fetch" value in the PAGE_FETCH_PAIRS column of any index (assuming this statistic is relevant for the index).
5. CARD must not be less than or equal to any "Fetch" value in the PAGE_FETCH_PAIRS column of any index (assuming this statistic is relevant for the index).

Rules for Updating Column Statistics

When you are updating statistics in SYSSTAT.COLUMNS, follow the guidelines below. For details on updating column distribution statistics, see “Rules for Updating Distribution Statistics for Columns” on page 333.

1. HIGH2KEY and LOW2KEY (in SYSSTAT.COLUMNS) must adhere to the following rules:
   - The datatype of any HIGH2KEY, LOW2KEY value must correspond to the datatype of the user column for which the statistic is attributed. Because HIGH2KEY is a VARCHAR column, you must enclose the value in quotation marks. For example, to set HIGH2KEY to 25 for an INTEGER user column, your update statement would include SET HIGH2KEY = '25'.
   - The length of HIGH2KEY, LOW2KEY values must be the smaller of 33 or the maximum length of the target column's datatype

- HIGH2KEY must be greater than LOW2KEY whenever there are 3 or more distinct values in the corresponding column. In the case of less than 3 distinct values in the column, HIGH2KEY can be equal to LOW2KEY.

2. The cardinality of a column (COLCARD statistic in SYSSTAT.COLUMNS) cannot be greater than the cardinality of its corresponding table (CARD statistic in SYSSTAT.TABLES).
3. No statistics are supported for columns with datatypes: LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB.

Rules for Updating Distribution Statistics for Columns

"User Update-Capable Catalog Statistics" on page 330 provides general information about how to update catalog statistics. You may wish to refer to that section before attempting to update column distribution statistics.

In order for all the statistics in the catalog to be consistent, you must exercise care when updating the distribution statistics. Specifically, for each column, the catalog entries for the frequent data statistics and quantiles must satisfy the following constraints:

1. Frequent value statistics (in the SYSSTAT.COLDIST catalog)
   - The values in column VALCOUNT must be non-increasing for increasing values of SEQNO
   - The number of values in column COLVALUE must be less than or equal to the number of distinct values in the column, which is stored in column COLCARD in catalog view SYSSTAT.COLUMNS.
   - The sum of the values in column VALCOUNT must be less than or equal to the number of rows in the column, which is stored in column CARD in catalog view SYSSTAT.TABLES
   - In most cases, the values in the column COLVALUE should lie between the second-highest and second-lowest data values for the column, which are stored in columns HIGH2KEY and LOW2KEY, respectively, in catalog view SYSSTAT.COLUMNS. There may be one frequent value greater than HIGH2KEY and one frequent value less than LOW2KEY.
2. Quantiles (in the SYSSTAT.COLDIST catalog)
   - The values in column COLVALUE must be non-decreasing for increasing values of SEQNO
   - The values in column VALCOUNT must be strictly increasing for increasing values of SEQNO
   - The largest value in column COLVALUE must have a corresponding entry in column VALCOUNT equal to the number of rows in the column
   - In most cases, the values in the column COLVALUE should lie between the second-highest and second-lowest data values for the column, which are stored in columns HIGH2KEY and LOW2KEY, respectively, in catalog view SYSSTAT.COLUMNS.

Suppose that distribution statistics are available for a column C1 with "R" rows and you wish to modify the statistics to correspond to a column with the same relative proportions of data values, but with "(F x R)" rows. To scale up the frequent-value

statistics by a factor of F, each entry in column VALCOUNT must be multiplied by F. Similarly, to scale up the quantiles by a factor of F, each entry in column VALCOUNT must be multiplied by F. If these rules are not followed, the optimizer may use the wrong filter factor causing unpredictable performance when you run the query.

Rules for Updating Index Statistics

When you update the statistics in SYSSTAT.INDEXES, follow the rules described below:

1. PAGE_FETCH_PAIRS (in SYSSTAT. INDEXES) must adhere to the following rules:

   - Individual values in the PAGE_FETCH_PAIRS statistic must be separated by a series of blank delimiters.
   - Individual values in the PAGE_FETCH_PAIRS statistic must not be longer than 10 digits and must be less than the maximum integer value (MAXINT = 2147483647).
   - There must always be a valid PAGE_FETCH_PAIRS value if the CLUSTERFACTOR is greater than zero.
   - There must be exactly 11 pairs in a single PAGE_FETCH_PAIR statistic.
   - Buffer size entries of PAGE_FETCH_PAIRS must be ascending in value.
   - If the buffer size value is the same as that in the previous pair, the page fetch value must be the same as that in the previous pair.
   - Any buffer size value in a PAGE_FETCH_PAIRS entry cannot be greater than MIN(NPAGES, 524287) where NPAGES is the number of pages in the corresponding table (in SYSSTAT.TABLES).
   - “Fetches” entries of PAGE_FETCH_PAIRS must be descending in value, with no individual “Fetches” entry being less than NPAGES. “Fetch” size values in a PAGE_FETCH_PAIRS entry cannot be greater than the CARD (cardinality) statistic of the corresponding table.
   - If buffer size value is the same in two consecutive pairs, then page fetch value must also be the same in both the pairs (in SYSSTAT.TABLES).

   A valid PAGE_FETCH_UPDATE is:

   ```
   PAGE_FETCH_PAIRS =
    '100 380 120 360 140 340 160 330 180 320 200 310 220 305 240 300
     260 300 280 300 300 300'
   ```

   where

   ```
   NPAGES = 300
   CARD   = 10000
   CLUSTERRATIO  =  -1
   CLUSTERFACTOR = 0.9
   ```

2. CLUSTERRATIO and CLUSTERFACTOR (in SYSSTAT.INDEXES) must adhere to the following rules:

   - Valid values for CLUSTERRATIO are -1 or between 0 and 100.
   - Valid values for CLUSTERFACTOR are -1 or between 0 and 1.
   - At least one of the CLUSTERRATIO and CLUSTERFACTOR values must be -1 at all times.

- If CLUSTERFACTOR is a positive value, it must be accompanied by a valid PAGE_FETCH_PAIR statistic.

3. The following rules apply to FIRSTKEYCARD, FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, and FULLKEYCARD:
   - FIRSTKEYCARD must be equal to FULLKEYCARD for a single-column index.
   - FIRSTKEYCARD must be equal to COLCARD for the corresponding column.
   - If any of these index statistics are not relevant, you should set them to -1. For example, if you have an index with only 3 columns, set FIRST4KEYCARD to -1.
   - For multiple column indexes, if all the statistics are relevant, the relationship between them must be:

     ```
     FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
        <= FULLKEYCARD <= CARD
     ```

4. The following rules apply to SEQUENTIAL_PAGES and DENSITY:
   - Valid values for SEQUENTIAL_PAGES are -1 or between 0 and NLEAF.
   - Valid values for DENSITY are -1 or between 0 and 100.

Updating Statistics for User-Defined Functions

Using the SYSSTAT.FUNCTIONS catalog view, you may update statistics for user-defined functions (UDFs). If these statistics are available, the optimizer will use them when estimating costs for various access plans. If statistics are not available the statistic column values will be -1 and the optimizer will use default values that assume a simple UDF.

The following table provides information about the statistic columns that you may update for UDFs:

Table 33 (Page 1 of 2). Function Statistics (SYSCAT.FUNCTIONS and SYSSTAT.FUNCTIONS)

Statistic	Description
IOS_PER_INVOC	Estimated number of read/write requests executed each time a function is executed.
INSTS_PER_INVOC	Estimated number of machine instructions executed each time a function is executed.
IOS_PER_ARGBYTE	Estimated number of read/write requests executed per input argument byte.
INSTS_PER_ARGBYTES	Estimated number of machine instructions executed per input argument byte.
PERCENT_ARGBYTES	Estimated average percent of input argument bytes that the function will actually process.
INITIAL_IOS	Estimated number of read/write requests executed only the first/last time the function is invoked.
INITIAL_INSTS	Estimated number of machine instructions executed only the first/last time the function is invoked.

Table 33 (Page 2 of 2). Function Statistics (SYSCAT.FUNCTIONS and SYSSTAT.FUNCTIONS)	
Statistic	**Description**
CARDINALITY	Estimated number of rows generated by a table function.

For example, consider a UDF (EU_SHOE) that converts an American shoe size to the equivalent European shoe size. (These two shoe sizes could be UDTs.) For this UDF, you should set the statistic columns as follows:

- INSTS_PER_INVOC should be set to the estimated number of machine instructions required to:
  - Invoke EU_SHOE
  - Initialize the output string
  - Return the result.
- INSTS_PER_ARGBYTE should be set to the estimated number of machine instructions required to convert the input string into a European shoe size.
- PERCENT_ARGBYTES would be set to 100 indicating that the entire input string is to be converted
- INITIAL_INSTS, IOS_PER_INVOC, IOS_PER_ARGBYTE, and INITIAL_IOS should all be set to 0, since this UDF only performs computations.

PERCENT_ARGBYTES would be used by a function that does not always process the entire input string. For example, consider a UDF (LOCATE) that accepts two arguments as input and returns the starting position of the first occurrence of the first argument within the second argument. Assume that the length of the first argument is small enough to be insignificant relative to the second argument and, on average, 75 percent of the second argument is searched. Based on this information, PERCENT_ARGBYTES should be set to 75. The above estimate of the average of 75 percent is based on the following additional assumptions:

- Half the time the first argument will not be found resulting in the entire second argument being searched
- The first argument is equally likely to appear anywhere within the second argument, resulting in half of the second argument being searched (on average) when the first argument is found.

INITIAL_INSTS or INITIAL_IOS can be used to record the estimated number of machine instructions or read/write requests performed only the first or last time the function is invoked. This could be used, for example, to record the cost of setting up a scratchpad area.

To obtain information about I/Os and instructions used by a user-defined function, you can use output provided by your programming language compiler or by monitoring tools available for your operating system.

Modelling Production Databases

Sometimes you may wish to have your test system contain a subset of your production system's data. However, access plans selected for such a test system are not necessarily the same as those that would be selected on the production system, unless the catalog statistics and the configuration parameters for the test system are updated to match those of the production system.

A productivity tool, `db2look`, is provided that can be run against the production database to generate the update statements required to make the catalog statistics of the test database match those in production. These update statements can be generated by using `db2look` in mimic mode (`-m` option). In this case, `db2look` will generate a command processor script containing all the statements required to mimic the catalog statistics of the production database.

You can recreate database data objects, including tables and indexes, by extracting DDL statements with `db2look -e`. You can run the command processor script created from this command against another database to recreate the database. You can use the `-e` option with the `-m` option.

After running the update statements produced by `db2look`, against the test system, the test system can be used to validate the access plans to be generated in production. Since the optimizer uses the type and configuration of the table spaces to estimate I/O costs, the test system must have the same table space geometry or layout. That is, the same number of containers of the same type: either SMS or DMS.

For more information on how to use this productivity tool, type the following on a command line:

```
db2look -h
```

You can also see the *Command Reference* manual.

Chapter 12. Understanding the SQL Compiler

When an SQL query is compiled, a number of steps are performed before the "best" access plan is either executed or written to the system catalog tables containing information about application packages.

In a partitioned database environment, all of the work done on a SQL query by the SQL Compiler takes place at the database partition to which you connect. Once the executable access plan is created, the compiled query is distributed to all database partitions in the database.

The following topics provide more information about the steps performed by the SQL Compiler:

- Overview of the SQL Compiler
- Query Rewrite by the SQL Compiler
- Operation Merging
- Operation Movement
- Predicate Translation
- Data Access Concepts and Optimization
- Optimization Strategies for Intra-partition Parallelism.

The following sections also provide information about factors external to the compiler which can affect the results produced by the compiler:

- Chapter 9, "Application Considerations" on page 265
- Chapter 10, "Environmental Considerations" on page 301
- Chapter 11, "System Catalog Statistics" on page 311.

Chapter 13, "SQL Explain Facility" on page 377 describes how you can examine the access plan chosen by the SQL compiler.

Overview of the SQL Compiler

The SQL compiler performs several steps before producing an access plan that you can execute. These steps are shown in Figure 39 on page 340.

This diagram shows that the **Query Graph Model** is a key component of the SQL compiler. The *query graph model* is an internal, in-memory database that is used to represent the query throughout the query compilation process as described below:

- **Parse Query**

  The first task of the SQL compiler is to analyze the SQL query to validate the syntax. If any syntax errors are detected, the SQL compiler stops processing and the appropriate SQL error is returned to the application attempting to compile the SQL statement. When parsing is complete, an internal representation of the query is created.

- **Check Semantics**

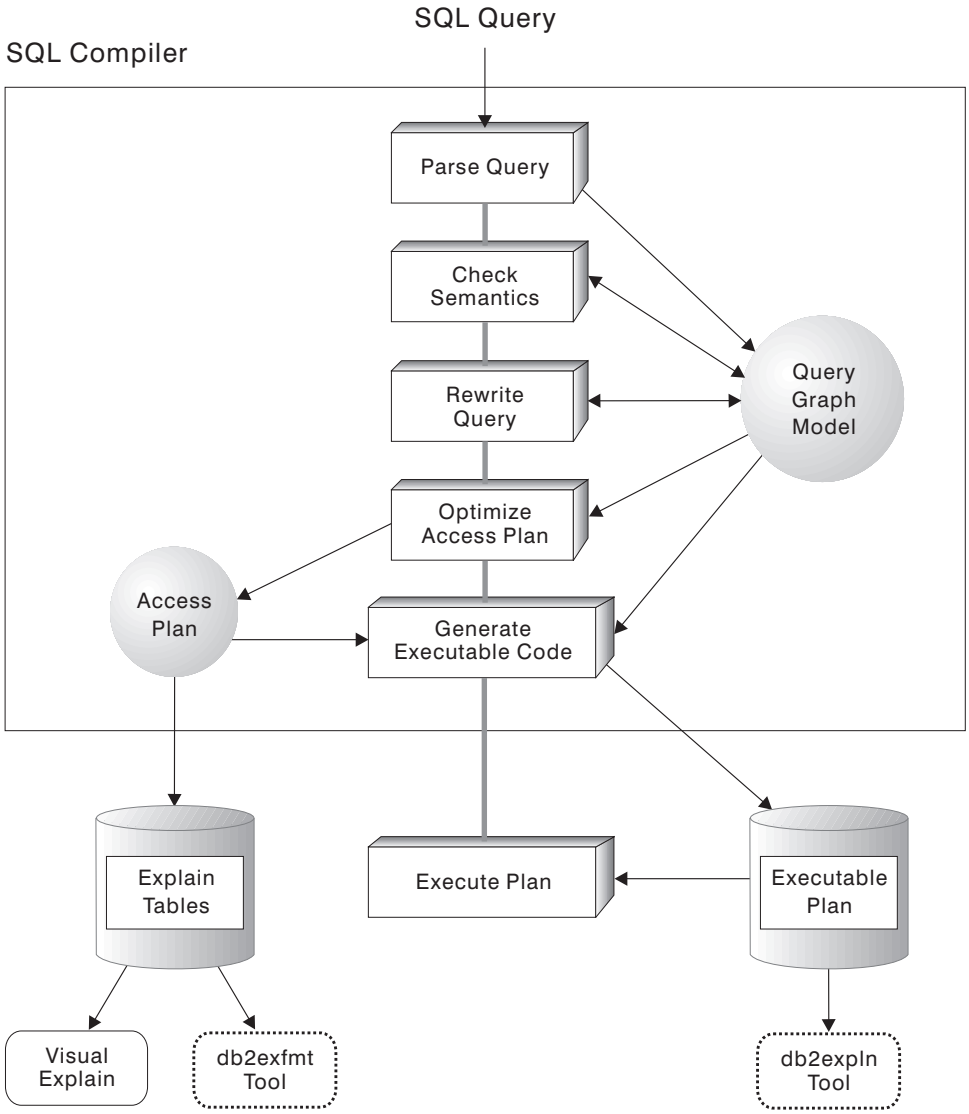


Figure 39. Steps performed by SQL Compiler

The second task of the compiler is to further validate the SQL statement by checking to ensure that the parts of the statement make sense given the other parts. A simple example of this semantic checking ensures that the data type of the column specified for the YEAR scalar function is a datetime data type. Also during this second stage, the compiler adds the behavioral semantics to the query graph model, including the effects of referential constraints, table check constraints, triggers, and views.

The query graph model contains all of the semantics of queries, including query blocks, subqueries, correlations, derived tables, expressions, data types, data type conversions, code page conversions, and partitioning keys.

- **Rewrite Query**

  The third phase of the SQL compiler uses the global semantics provided in the query graph model to transform the query into a form that can be optimized more easily. See “Query Rewrite by the SQL Compiler” on page 342 for more information.

  Working in a partitioned database environment, some query operations are more computationally intensive like those involving:

  – Aggregation
  – Redistribution of rows
  – Correlated subqueries.

  In this environment, with some queries, decorrelation can occur as part of the rewrite of the query.

  Any transformations that occur on a query are written back to the query graph model. That is, the query graph model represents the rewritten query.

- **Optimize Access Plan**

  The SQL optimizer portion of the SQL compiler uses the query graph model as input, and generates many alternative execution plans for satisfying the user's request. It estimates the execution cost of each alternative plan, using the statistics for tables, indexes, columns and functions, and chooses the plan with the smallest estimated execution cost. The optimizer uses the query graph model to analyze the query semantics and to obtain information about a wide variety of factors, including indexes, base tables, derived tables, subqueries, correlations and recursion. That the environment includes a partitioned database is also considered as well as the ability to enhance the chosen plan for the possibility of intra-query parallelism in a symmetric multi-processor (SMP) environment. This information is used by the optimizer to help select the best access plan for the query. See “Data Access Concepts and Optimization” on page 349 for more information.

  The output from this step of the SQL compiler is an “access plan.” This access plan provides the basis for the information captured in the Explain tables. The information used to generate the access plan can be captured with an explain snapshot. (See Chapter 13, “SQL Explain Facility” on page 377 for more information on Explain topics.)

- **Generate “Executable” Code**

  The final step of the SQL Compiler uses the *access plan* and the query graph model to create an executable access plan, or section, for the query. This code generation step uses information from the query graph model to avoid repetitive execution of expressions that only need to be computed once for a query. Examples for which this optimization is possible include code page conversions and the use of host variables.

Information about access plans for static SQL is stored in the system catalog tables. When the package is executed, the database manager will use the information stored in the system catalog tables to determine how to access the data and provide results for the query. It is this information that is used by the **db2expln** tool. (See Chapter 13, "SQL Explain Facility" on page 377 for more information on Explain topics.)

It is recommended that RUNSTATS be done periodically on tables used in queries where good performance is desired. The optimizer will then be better equipped with relevant statistical information on the nature of the data. If RUNSTATS is not done (or the optimizer suspects that RUNSTATS was done on empty or near empty tables), the optimizer may either use defaults or attempt to derive certain statistics based on the number of file pages used to store the table on disk (FPAGES).

Query Rewrite by the SQL Compiler

The SQL compiler includes a query rewrite stage which transforms SQL statements into forms that can be optimized more easily, and as a result, can improve the access path chosen. Rewriting queries is particularly important for queries which are very complex, including those queries with many subqueries or many joins. Query generator tools often create these types of very complex queries.

You can influence the number of query rewrite rules that are applied to an SQL statement by changing the optimization class (see "Adjusting the Optimization Class" on page 283).

You can see some of the results of the query rewrite through the use of the Explain facility or Visual Explain.

There are three major categories of rewriting that the SQL compiler may perform:

- Operation Merging
- Operation Movement
- Predicate Translation.

Operation Merging

The SQL compiler will rewrite queries to merge query operations, in an attempt to construct the query so that it has a single SELECT operation. The following examples are provided to illustrate some of the operations that can be merged by the SQL compiler:

- Example - View Merges

  Using views in a SELECT statement can restrict the join order of the table and can also introduce redundant joining of tables. By merging the views during query rewrite, these restrictions can be lifted.

- Example - Subquery to Join Transformations

The use of subqueries in a SELECT statement can force a join method and the selection of inner and outer tables for the join. During query rewrite, the subquery can sometimes be merged into the main query as a join, which gives the optimizer more choices to choose the most efficient access plan.

- Example - Redundant Join Elimination

  During query rewrite redundant joins can be removed to further simplify the SELECT statement that will be optimized.

- Example - Shared Aggregation

  When using different functions, rewriting the query can reduce the number of calculations that need to be done.

Example - View Merges

Suppose you have access to the following two views of the EMPLOYEE table, one showing employees with a high level of education and the other view showing employees earning more than $35,000:

```
CREATE VIEW EMP_EDUCATION (EMPNO, FIRSTNME, LASTNAME, EDLEVEL) AS
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL
  FROM EMPLOYEE
 WHERE EDLEVEL > 17
CREATE VIEW EMP_SALARIES (EMPNO, FIRSTNAME, LASTNAME, SALARY) AS
SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
  FROM EMPLOYEE
 WHERE SALARY > 35000
```

Now suppose you perform the following query to list the employees who have a high education level and who are earning more than $35,000:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMP_EDUCATION E1,
       EMP_SALARIES  E2
 WHERE E1.EMPNO = E2.EMPNO
```

During query rewrite, these two views could be merged to create the following query:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

By merging the SELECT statements from the two views with the user-written SELECT statement, the optimizer can consider more choices when selecting an access plan. In addition, if the two views that have been merged use the same base table, additional rewriting may be performed as described in “Example - Redundant Join Elimination” on page 344.

Example - Subquery to Join Transformations

The SQL compiler will take a query containing a subquery, such as:

```
SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO
FROM EMPLOYEE
WHERE WORKDEPT IN
      (SELECT DEPTNO
         FROM DEPARTMENT
        WHERE DEPTNAME = 'OPERATIONS')
```

and convert it to a join query of the form:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME, PHONENO
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNAME = 'OPERATIONS'
```

A join is generally much more efficient to execute than a subquery.

Example - Redundant Join Elimination

Queries can sometimes be written or generated which have unnecessary joins. Queries such as the following could also be produced during the query rewrite stage as described in “Example - View Merges” on page 343.

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

In this query, the SQL compiler can eliminate the join and simplify the query to:

```
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL, SALARY
  FROM EMPLOYEE
 WHERE EDLEVEL > 17
   AND SALARY  > 35000
```

Example - Shared Aggregation

Using multiple functions within a query can generate several calculations which take time. Reducing the number of calculations to be done within the query results in an improved plan. The SQL compiler takes a query using multiple functions such as:

```
SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  AVG(SALARY+BONUS+COMM) AS OAVG,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE;
```

and transforms the query in the following way:

```
SELECT OSUM,
  OSUM/OCOUNT
  OCOUNT
FROM (SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE) AS SHARED_AGG;
```

This rewrite reduces the query from 2 sums and 2 counts to 1 sum and 1 count.

Operation Movement

The SQL compiler will rewrite queries to move query operations in an attempt to construct the query with the minimum number of operations and predicates. The following examples are provided to illustrate some of the operations that can be moved by the SQL compiler:

- Example - DISTINCT Elimination

  During query rewrite, the optimizer can move where the DISTINCT operation is performed, to reduce the cost of this operation. In the example provided, the DISTINCT operation is removed completely.

- Example - General Predicate Pushdown

  During query rewrite, the order of applying predicates can be changed so that more selective predicates are applied to the query as early as possible.

- Example - Decorrelation

  When in a partitioned database environment the movement of results sets between database partitions is costly. Reducing the size of what must be broadcast to other database partitions and/or the number of broadcasts is one of the objectives when rewriting queries.

Example - DISTINCT Elimination

If the EMPNO column was defined as the primary key of the EMPLOYEE table, the following query:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME
  FROM EMPLOYEE
```

would be rewritten by removing the DISTINCT clause:

```
SELECT EMPNO, FIRSTNME, LASTNAME
  FROM EMPLOYEE
```

In the above example, since the primary key is being selected, the SQL compiler knows that each row returned will already be unique. In this case, the DISTINCT key word is redundant. If the query was not rewritten, the optimizer would build a plan with the necessary processing (a sort, for example) to ensure that the columns are distinct.

Example - General Predicate Pushdown

Altering the level at which a predicate is normally applied can result in improved performance. For example, given the following view which provides a list of all employees in department “D11”:

```
CREATE VIEW D11_EMPLOYEE
 (EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM)
AS SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM
     FROM EMPLOYEE
    WHERE WORKDEPT = 'D11'
```

And given the following query:

```
SELECT FIRSTNME, PHONENO
  FROM D11_EMPLOYEE
 WHERE LASTNAME = 'BROWN'
```

The query rewrite stage of the compiler will push the predicate `LASTNAME = 'BROWN'` up into the view D11_EMPLOYEE. This allows the predicate to be applied sooner and potentially more efficiently. The actual query that could be executed in this example is:

```
SELECT FIRSTNME, PHONENO
  FROM EMPLOYEE
 WHERE LASTNAME = 'BROWN'
   AND WORKDEPT = 'D11'
```

Pushdown of predicates is not limited to views. Other situations in which predicates may be pushed down include UNIONs, GROUP BYs, and derived tables (nested table expressions or common table expressions).

Example - Decorrelation

In a partitioned database environment, the SQL compiler can rewrite the following query:

Find all the employees who are working on programming projects and are underpaid.

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
  FROM EMPLOYEE E, PROJECT P
 WHERE P.EMPNO = E.EMPNO
   AND P.PROJNAME LIKE '%PROGRAMMING%'
   AND E.SALARY+E.BONUS+E.COMM <
     (SELECT AVG(E1.SALARY+E1.BONUS+E1.COMM)
        FROM EMPLOYEE E1, PROJECT P1
        WHERE P1.PROJNAME LIKE '%PROGRAMMING%'
          AND P1.PROJNO = A.PROJNO
          AND E1.EMPNO = P1.EMPNO)
```

Since this query is correlated, and since both PROJECT and EMPLOYEE are unlikely to be partitioned on PROJNO, the broadcast of each project to each database partition is possible. In addition, the subquery would have to be evaluated many times.

The SQL compiler can rewrite the query as follows:

- Determine the distinct list of employees working on programming projects, DIST_PROJS, otherwise we'll aggregate on non-distinct project numbers multiple times, yielding incorrect results:

```
WITH DIST_PROJS(PROJNO, EMPNO) AS
(SELECT DISTINCT PROJNO, EMPNO
 FROM PROJECT P1
 WHERE P1.PROJNAME LIKE '%PROGRAMMING%')
```

- Using the distinct list of employees working on the programming projects, join this to the employee table, to get the average compensation per project, AVG_PER_PROJ:

```
AVG_PER_PROJ(PROJNO, AVG_COMP) AS
(SELECT P2.PROJNO, AVG(E1.SALARY+E1.BONUS+E1.COMM)
 FROM EMPLOYEE E1, DIST_PROJS P2
 WHERE E1.EMPNO = P2.EMPNO
 GROUP BY P2.PROJNO)
```

- Then the new query would be:

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
 FROM PROJECT P, EMPLOYEE E, AVG_PER_PROG A
WHERE P.EMPNO = E.EMPNO
  AND P.PROJNAME LIKE '%PROGRAMMING%'
  AND P.PROJNO = A.PROJNO
  AND E.SALARY+E.BONUS+E.COMM < A.AVG_COMP
```

The rewritten SQL query computes the AVG_COMP per project (AVG_PRE_PROJ) and can then broadcast the result to all database partitions containing the EMPLOYEE table.

Predicate Translation

The SQL compiler will rewrite queries to translate existing predicates to more optimal predicates for the specific query. The following examples are provided to illustrate some of the predicates that could be translated by the SQL compiler:

- Example - Addition of Implied Predicates

  During query rewrite, predicates can be added to the query to allow the optimizer to consider additional table joins when selecting the best access plan for the query.

- Example - OR to IN Transformations

  During query rewrite, an OR predicate can be translated into an IN predicate to allow for a more efficient access plan to be chosen. The SQL compiler can also translate an IN predicate into an OR predicate if this transformation would allow a more efficient access plan to be chosen.

Example - Addition of Implied Predicates

The following query produces a list of the managers whose departments report to “E01” and the projects for which those managers are responsible:

```
SELECT DEPT.DEPTNAME DEPT.MGRNO, EMP.LASTNAME, PROJ.PROJNAME
  FROM DEPARTMENT DEPT,
       EMPLOYEE   EMP,
       PROJECT    PROJ
 WHERE DEPT.ADMRDEPT = 'E01'
   AND DEPT.MGRNO = EMP.EMPNO
   AND EMP.EMPNO  = PROJ.RESPEMP
```

The query rewrite will add the following implied predicate:

```
DEPT.MGRNO = PROJ.RESPEMP
```

As a result of this rewrite, the optimizer can consider additional joins when it is trying to select the best access plan for the query.

In addition to the above predicate transitive closure, query rewrite will also derive additional local predicates based on the transitivity implied by equality predicates. For example, the following query lists the names of the departments (whose department number is greater than “E00”) and employees who work in that department.

```
SELECT EMPNO, LASTNAME, FIRSTNAME, DEPTNO, DEPTNAME
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNO > 'E00'
```

For this query, the rewrite stage will add the following implied predicate:

```
EMP.WORKDEPT > 'E00'
```

As a result of this rewrite, the optimizer reduces the number of rows to be joined.

Example - OR to IN Transformations

Suppose an OR clause connects two or more simple equality predicates on the same column, as in the following example:

```
SELECT *
  FROM EMPLOYEE
 WHERE DEPTNO = 'D11'
    OR DEPTNO = 'D21'
    OR DEPTNO = 'E21'
```

If there is no index on the DEPTNO column, converting the OR clause to the following IN predicate will allow the query to be processed more efficiently:

```
SELECT *
  FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

Note: In some cases, the database manager may convert an IN predicate to a set of OR clauses so that index ORing may be performed. See “Multiple Index Access” on page 354 for more information about index ORing.

Data Access Concepts and Optimization

When compiling an SQL statement, the SQL optimizer estimates the execution cost of different ways of satisfying your request. Based on this evaluation, the optimizer selects what it believes to be the optimal access plan. An *access plan* specifies the order of operations required to resolve an SQL statement. When an application program is bound, a *package* is created. This package contains access plans for all of the static SQL statements in that application program. Access plans for dynamic SQL statements are created at the time that the application is executed.

There are two ways of accessing data in a table: by directly reading the table (relation scan), or by first accessing an index on that table (index scan).

A *relation scan* occurs when the database manager sequentially accesses every row of a table. See “Index Scan Concepts” to learn how an index scan works and see “Relation Scan versus Index Scan” on page 356 to understand under what conditions each type of scan is used.

The following topics describe other methods that can also be used in an access plan to access data in a table, and to produce the results for your query:

Other Related Topics:

- “Adjusting the Optimization Class” on page 283, provides information about controlling the number of alternative access plans evaluated by the SQL compiler
- Chapter 13, “SQL Explain Facility” on page 377, provides information about how you can obtain information about the access plan chosen by the SQL compiler.

Index Scan Concepts

An *index scan* occurs when the database manager accesses an index to do any or all of the following:

- Narrow down the set of qualifying rows (by scanning the rows in a certain range of the index) before accessing the base table. The index *scan range* (the start and stop points of the scan) is determined by the values in the query against which index columns are being compared.
- Order the output.
- Fully retrieve the requested data. If all of the requested data is in the index, the base table will not be accessed. This is known as an *Index-only access*.

The following additional topics are provided:

- Index Structure
- Index Scans to Delimit a Range
- Index Scans to Order Data
- Index-Only Access
- Multiple Index Access
- Index Clustering
- Index Page Prefetch

Index Structure

The database manager uses a B+ tree structure for storing its indexes. A B+ tree has one or more levels, as shown in the following diagram (where `RID` means row ID):

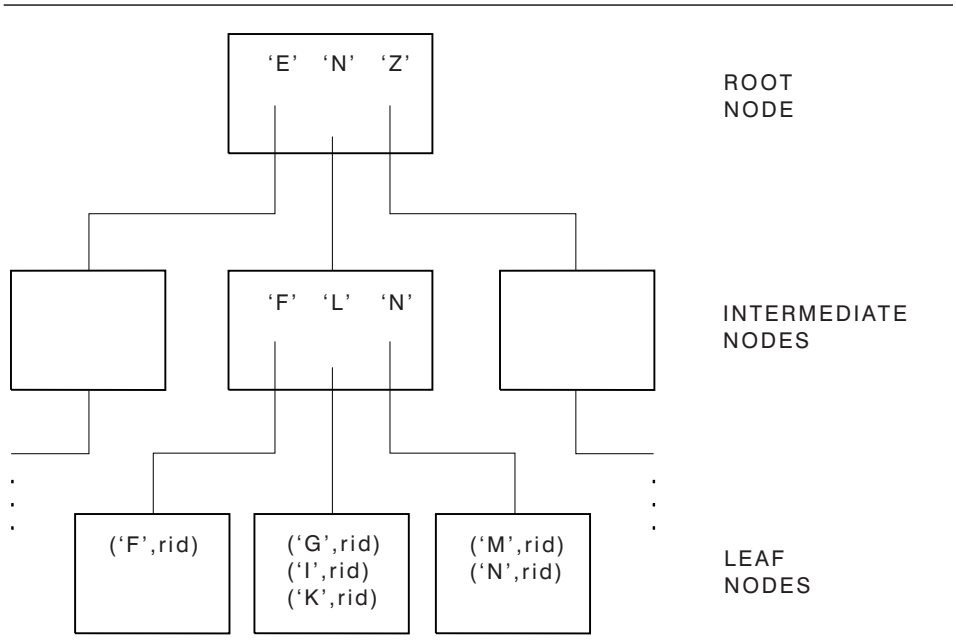


Figure 40. B+ Tree Structure

The top level is called the *root node.* The bottom level consists of *leaf nodes*, where the actual index key values are stored, as well as a pointer to the actual row in the table. Levels between the root and leaf node levels are called *intermediate nodes*.

In looking for a particular index key value, Index Manager searches the index tree, starting at the root node. The root contains one key for each node at the next level. The value of each of these keys is the largest existing key value for the corresponding node at the next level. For example, if an index has three levels as shown in Figure 40, then to find an index key value, Index Manager would search the root node for the first key value greater than or equal to the key being looked for. This root node key would point to a specific intermediate node. The same procedure would be followed with that intermediate node to determine which leaf node to go to. The final index key would be

found in the leaf node. Using Figure 40, the key being looked for is “I.” The first key in the root node greater than or equal to “I” is “N.” This points to the middle node at the next level. The first key in that intermediate node that is greater than or equal to “I” is “L.” This points to a specific leaf node where the index key for “I” along with its corresponding row ID(s) are found (the row ID of the corresponding rows in the base table).

Index Scans to Delimit a Range

In determining whether an index can be used for a particular query, the optimizer evaluates each column of the index starting with the first column to see if it can be used to satisfy:

- Any of the EQUAL predicates in the statement's WHERE clause
- Any other predicates in the WHERE clause.

A *predicate* is an element of a search condition in a WHERE clause that expresses or implies a comparison operation. Predicates that can be used to delimit the range of an index scan are those involving an index column in which one of the following is true:

- The index column is being tested for equality against a constant, a host variable, an expression that evaluates to a constant, or a keyword
- The test against the index column is “IS NULL” or “IS NOT NULL”
- The test is for equality against a basic subquery (that is, one that does not contain ANY, ALL, or SOME), and the subquery does not have a correlated column reference to its immediate parent query block (that is, the SELECT for which this subquery is a subselect).
- The test is an inequality predicate meeting the conditions described below.

For example, given an index with the following definition:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

the following predicates could be used in delimiting the range of the scan of index IX1:

```
WHERE  NAME = :hv1
  AND  DEPT = :hv2
```

or

```
WHERE  MGR  = :hv1
  AND  NAME = :hv2
  AND  DEPT = :hv3
```

Note that in the second example the WHERE predicates do not have to be specified in the same order as the key columns appear in the index. And, although host variables are used in the examples, parameter markers, expressions, or constants would have the same effect.

In the following WHERE clause, only the predicates for NAME and DEPT would be used in delimiting the range of the index scan, but not the predicates for SALARY or YEARS:

```
WHERE  NAME   = :hv1
  AND  DEPT   = :hv2
  AND  SALARY = :hv4
  AND  YEARS  = :hv5
```

This is because there is a key column (MGR) separating these columns from the first two index key columns, so the ordering would be off. However, once the range is determined by the NAME = :hv1 and DEPT = :hv2 predicates, the remaining predicates can be evaluated against the remaining index key columns.

In addition to the equality predicates described above, certain inequality predicates may be used to delimit the range of an index scan. The following discusses the two types of inequality predicates: strict inequality and inclusive inequality.

Strict Inequality Predicates: The strict inequality operators which can be used for range delimiting predicates are > and <.

For delimiting a range for an index scan, only one column with strict inequality predicates will be considered. In the following example, the predicates on the NAME and DEPT columns can be used to delimit the range, but the predicate on the MGR column cannot be used.

```
WHERE  NAME   = :hv1
  AND  DEPT   > :hv2
  AND  DEPT   < :hv3
  AND  MGR    < :hv4
```

Inclusive Inequality Predicates: The following are inclusive inequality operators which can be used for range delimiting predicates:

- >= and <=
- BETWEEN
- LIKE

For delimiting a range for an index scan, multiple columns with inclusive inequality predicates will be considered. In the following example, all of the predicates can be used to delimit the range of the index scan:

```
WHERE  NAME   = :hv1
  AND  DEPT  >= :hv2
  AND  DEPT  <= :hv3
  AND  MGR   <= :hv4
```

To further illustrate this example, suppose that `:hv2 = 404`, `:hv3 = 406`, and `:hv4 = 12345`. The database manager will scan the index for all of departments 404 and 405, but it will stop scanning department 406 when it reaches the first manager that has an employee number (MGR column) greater than 12345.

For additional information, see “Range Delimiting and Index SARGable Predicates” on page 357.

Index Scans to Order Data

If the query involves ordering, an index can be used to order the data if the ordering columns appear consecutively in the index, starting from the first index key column. (Ordering or sorting can result from operations such as ORDER BY, DISTINCT, GROUP BY, “= ANY” subquery, “> ALL” subquery, “< ALL” subquery, INTERSECT or EXCEPT, UNION.) An exception to this is when the index key columns are compared for equality against “constant values” (that is, any expression that evaluates to a constant). In this case the ordering column can be other than the first index key columns. For example, in the query:

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY MGR
```

the index could be used to order the rows since NAME and DEPT will always be the same values and will thus be ordered. Another way of saying this is that the preceding WHERE and ORDER BY clauses are equivalent to:

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY NAME, DEPT, MGR
```

A unique index can also be used to truncate an order requirement. For example, given the following index definition and order by clause:

```
UNIQUE INDEX IX0:  PROJNO  ASC
SELECT PROJNO, PROJNAME, DEPTNO
  FROM PROJECT
ORDER BY PROJNO, PROJNAME
```

additional ordering on the PROJNAME column is not required since the IX0 index ensures that PROJNO is unique. This uniqueness ensures that there is only one PROJNAME value for each PROJNO value.

Index-Only Access

In some cases, all of the required data can be retrieved from the index without accessing the table. This is known as an *index-only* access.

To illustrate an index-only access, consider the following index definition:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

and the following query can be satisfied by accessing only the index, and without reading the base table:

```
SELECT NAME, DEPT, MGR, SALARY
  FROM EMPLOYEE
 WHERE NAME = 'SMITH'
```

In other cases, there may be columns that do not appear in the index. To obtain the data for these columns, rows of the base table must be read. For example, given the IX1 index, the following query needs to access the base table to obtain the PHONENO and HIREDATE column data:

```
SELECT NAME, DEPT, MGR, SALARY, PHONENO, HIREDATE
  FROM EMPLOYEE
  WHERE NAME = 'SMITH'
```

Multiple Index Access

In all of the above examples, a single index scan was performed to produce the results. To satisfy the predicates of a WHERE clause, the optimizer can choose to scan multiple indexes. For example, given the following two index definitions:

```
INDEX IX2:  DEPT    ASC
INDEX IX3:  JOB     ASC,
            YEARS   ASC
```

the following predicates could be resolved using these two indexes:

```
WHERE DEPT = :hv1
  OR (JOB   = :hv2
  AND YEARS >= :hv3)
```

In this example, scanning index IX2 will produce a list of row IDs (RIDs) that satisfy the `DEPT = :hv1` predicate. Scanning index IX3 will produce a list of RIDs satisfying the `JOB = :hv2 AND YEARS >= :hv3` predicate. These two lists of RIDs can be combined and duplicates removed before accessing the table. This is known as *index ORing*.

Index ORing may also be used for predicates using the IN expression, as in the following example:

```
WHERE DEPT IN (:hv1, :hv2, :hv3)
```

With index ORing you are looking to eliminate duplicate RIDs, however with *index ANDing* you are looking for RIDs that occur in every index scanned. Index ANDing may occur with applications where there are multiple indexes on corresponding columns within the same table and a query using multiple “and” predicates is run against that table. Multiple index scans against each indexed column in such a query produce qualifying rows that have their RID values hashed to dynamically create bitmaps. The second bitmap is used to probe the first bitmap to generate the qualifying rows that are fetched to create the final returned data set.

For example, given the following two index definitions:

```
INDEX IX4: SALARY   ASC
INDEX IX5: COMM     ASC
```

the following predicates could be resolved using these two indexes:

```
WHERE SALARY BETWEEN 20000 AND 30000
   AND COMM BETWEEN 1000 AND 3000
```

In this example, scanning index IX4 produces a dynamic bitmap index satisfying the SALARY BETWEEN 20000 AND 30000 predicate. Scanning IX5 and probing the dynamic bitmap index for IX4 results in the list of qualifying RIDs that satisfy both predicates. This is known as "dynamic bitmap ANDing." It only occurs if the table has sufficient cardinality and the columns have sufficient values in the qualifying range, or sufficient duplication if equality predicates are used.

Note: In the accessing of any single table, DB2 does not combine index ANDing and index ORing.

Index Clustering

When selecting the access plan, the optimizer considers the I/O cost of fetching pages from disk to the buffer pool. In its calculations, the optimizer will estimate the number of I/Os required to satisfy a query. This estimate includes a prediction of buffer pool usage, since additional I/Os are not required to read rows in a page that is already in the buffer pool.

For index scans, the optimizer uses information from the system catalog tables (SYSCAT.INDEXES) to help estimate I/O cost of reading data pages into the buffer pool. The following columns from the SYSCAT.INDEXES table are used:

- CLUSTERRATIO indicating the degree of clustering of the table data in relation to this index. Higher clustering of the data generally means that the rows are ordered on the data pages in index key sequence. Therefore, all of the rows on a data page can be read while the page is in buffer. If the value of this column is -1, the optimizer will attempt to use the CLUSTERFACTOR column.

  or

- CLUSTERFACTOR indicating the degree of clustering, but at a finer level than CLUSTERRATIO. When collecting statistics for an index, this information is considered a detailed statistic.
- PAGE_FETCH_PAIRS containing several pairs of numbers which model the number of I/Os required to read the data pages into buffer pools of various sizes. This is only used when CLUSTERFACTOR is available. When collecting statistics for an index, this information is considered a detailed statistic.

If statistics are not available, the optimizer will use default values for the statistics, which assume poor clustering of the data to the index. See also Chapter 11, "System Catalog Statistics" on page 311 and " Collecting Statistics using the RUNSTATS Utility" on page 312.

You can specify an index that will be used to cluster the rows during a table reorganization. (See "Reorganizing Table Data" on page 415 for information about table reorganization.) Subsequent updates and inserts may reduce the clustering of such an index, so you may need to periodically reorganize the table to maintain the clustering.

Clustering of the data can have a significant impact on performance and you should try to keep one of the indexes on the table close to 100 percent clustered.

Note: Only one index can be one hundred percent clustered, except in those cases where the keys are a superset of the keys of the clustered index; or, where there is de facto correlation between the key columns of the two indexes.

The above concept of clustering data during a table reorganization is not the same as the concept of a *clustering index*, which is available in some members of the DB2 family (for example, DB2 for MVS/ESA). In these other products, the clustering index identifies a single index per table, for which the database manager maintains row clustering in the tables, as much as possible, across inserts and updates.

Clustering Page Reads Using List Prefetch: If the optimizer uses an index to access rows, it can defer reading the data pages until all the RIDs (row identifiers) have been obtained from the index. For example, given the previously defined index IX1:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

and the following search criteria:

```
WHERE NAME BETWEEN 'A' and 'I'
```

the optimizer could perform an index scan on IX1 to determine the rows (and data pages) to retrieve. If the data was not clustered according to this index, list prefetch will include a step to sort the list of RIDs obtained from the index scan. See “Understanding List Prefetching” on page 407 for more information.

Index Page Prefetch

When appropriate, the database manager detects sequential access to index pages and will generate prefetch requests. This will significantly reduce the elapsed time for nonselective index scans, and selective index scans accessing a significant portion of the index.

The optimizer uses index statistics such as DENSITY and SEQUENTIAL_PAGES, the characteristics of the table spaces in which the index resides, and the effect of any range delimiting predicates, to estimate the amount of index page prefetch that will occur. These estimates are factored into the overall cost estimate for using a particular index.

See “Understanding Sequential Prefetching” on page 406 for more information.

Relation Scan versus Index Scan

The optimizer will choose a relation scan when an index cannot be used for the query, or if the optimizer determines that an index scan would be more costly. An index scan could be more costly when:

- The table is small
- Index clustering is low
- Most of the table is accessed.

You may use the SQL Explain facilities to determine whether your access plan uses a relation scan or an index scan. See Chapter 13, “SQL Explain Facility” on page 377.

Predicate Terminology

A user application requests a set of rows from the database with an SQL statement, qualifying the specific rows desired through the use of predicates. When the optimizer decides how to evaluate an SQL statement, each predicate falls into one of four categories. The category is determined by how and when that predicate is used in the evaluation process. These categories are listed below, ordered in terms of performance from best to worst:

1. Range delimiting predicates
2. Index SARGable predicates
3. Data SARGable predicates
4. Residual predicates.

SARGable refers to something that can be used as a *s*earch *arg*ument.

“Summary of Predicate Usage” on page 358 provides a comparison of the characteristics that affect the performance of the various predicate categories.

Range Delimiting and Index SARGable Predicates

Range delimiting predicates are those used to bracket an index scan. They provide start and/or stop key values for the index search. Index SARGable predicates are not used to bracket a search, but can be evaluated from the index because the columns involved in the predicate are part of the index key. For example, given the previously defined index IX1 (in the section “Index Scan Concepts” on page 349) and the following WHERE clause:

```
WHERE  NAME  = :hv1
 AND   DEPT  = :hv2
 AND   YEARS > :hv5
```

the first two predicates (NAME = :hv1, DEPT = :hv2) would be range delimiting predicates, while YEARS > :hv5 would be an index SARGable predicate.

The database manager will make use of the index data in evaluating these predicates rather than reading the base table. These *index SARGable* predicates reduce the number of data pages accessed by reducing the set of rows that need to be read from the table. These types of predicates do not affect the number of index pages that are accessed.

Data SARGable Predicates

Predicates that cannot be evaluated by Index Manager, but can be evaluated by Data Management Services are called *data SARGable* predicates. Typically, these

predicates require the access of individual rows from a base table. If required, Data Management Services will retrieve the columns needed to evaluate the predicate, as well as any others to satisfy the columns in the SELECT list that could not be obtained from the index.

For example, given a single index defined on the PROJECT table:

```
INDEX IX0:  PROJNO ASC
```

And given the following query, the `DEPTNO = 'D11'` predicate is considered to be data SARGable.

```
SELECT PROJNO, PROJNAME, RESPEMP
  FROM PROJECT
 WHERE DEPTNO = 'D11'
 ORDER BY PROJNO
```

Residual Predicates

Residual predicates, typically, are those that require I/O beyond the simple accessing of a base table. Examples of residual predicates include those using correlated subqueries, using quantified subqueries (subqueries with ANY, ALL, SOME, or IN), or reading LONG VARCHAR or LOB data (stored in a file separate from the table). These predicates are evaluated by Relational Data Services.

Sometimes predicates, which are applied to the index only, have to be reapplied when the data page is accessed. For example, access plans using index ORing or index ANDing, (see “Multiple Index Access” on page 354), always reapply the predicates as residual predicates, when the data page is accessed.

Summary of Predicate Usage

The use of predicates in a query can help to reduce the amount of data read to satisfy the query. Different categories of predicates have different impacts on the performance of a query and these impacts are considered by the optimizer. The following table shows the ranking of the different types of predicates and how each type of predicate can influence performance.

Table 34. Summary of Predicate Type Characteristics

Characteristic	Predicate Type			
	Range Delimiting	**Index SARGable**	**Data SARGable**	**Residual**
Reduce index I/O	Yes	No	No	No
Reduce data page I/O	Yes	Yes	No	No
Reduce number of rows passed internally	Yes	Yes	Yes	No
Reduce number of qualifying rows	Yes	Yes	Yes	Yes

Join Concepts

A *join* is where rows from one table are concatenated to rows of one or more other tables. For example, given the following two tables:

```
      TABLE1                    TABLE2
-----------------          -----------------
 PROJ     PROJ_ID          PROJ_ID     NAME
------    -------          -------    ------
  A          1                1        Sam
  B          2                3        Joe
  C          3                4        Mary
  D          4                1        Sue
                              2        Mike
```

Joining Table1 and Table2 where the ID columns are equal would be represented by the following SQL statement:

```
SELECT PROJ, x.PROJ_ID, NAME
  FROM TABLE1 x, TABLE2 y
  WHERE x.PROJ_ID = y.PROJ_ID
```

and would yield the following set of result rows:

```
 PROJ     PROJ_ID     NAME
------    -------    ------
  A          1        Sam
  A          1        Sue
  B          2        Mike
  C          3        Joe
  D          4        Mary
```

When joining two tables, one table is selected as the outer table and the other as the inner. The outer table is accessed first and is only scanned once. Whether the inner table is scanned multiple times depends on the type of join and which indexes are present. Whether your query joins two tables or more than two tables, the optimizer will only join two tables at a time. If needed, temporary, intermediary results tables will be created.

The optimizer will choose one of the two join methods (nested loop join or merge join) depending on the existence of a join predicate (defined in “Merge Join” on page 361), as well as various costs involved as determined by table and index statistics.

Nested Loop Join

A nested loop join is performed in one of two ways:

1. By scanning through the inner table for each accessed row of the outer table

   For example, if column A in tables T1 and T2 has the following values:

```
Outer Table T1: column A      Inner Table T2: column A
-----------------------       -----------------------
          2                             3
          3                             2
          3                             2
                                        3
                                        1
```

The steps for doing the nested loop:

- Read the first row from T1. The value for A is “2”
- Scan T2 until a match (“2”) is found, and then join the two rows
- Scan T2 until the next match (“2”) is found, and then join the two rows
- Scan T2 to the end of the table
- Go back to T1 and read the next row (“3”)
- Scan T2, starting at the first row, until a match (“3”) is found, and then join the two rows
- Scan T2 until the next match (“3”) is found, and then join the two rows
- Scan T2 to the end of the table
- Go back to T1 and read the next row (“3”)
- Scan T2 as before, joining all rows which match (“3”).

2. By doing an index lookup on the inner table for each accessed row of the outer table.

   This method can be used for the specified predicates if there is a predicate of the following form:

   ```
   expr(outer_table.column)  relop  inner_table.column
   ```

   where `relop` is a relative operator (for example =, >, >=, <, or <=) and `expr` is a valid expression on the outer table. The following are examples:

   ```
   OUTER.C1 + OUTER.C2 <= INNER.C1
   ```

   and

   ```
   OUTER.C4 < INNER.C3
   ```

   This method could be a way to significantly reduce the number of rows accessed in the inner table for each access of the outer table (although it depends on a number of factors, including the selectivity of the join predicate).

When evaluating a nested loop join, the optimizer will also determine whether or not to sort the outer table before performing the join. By ordering the outer table, based on the join columns, the number of read operations to access pages from disk for the inner table may be reduced, since it is more likely they will already be in the buffer pool. If the join uses a highly clustered index to access the inner table, the number of index pages accessed may be minimized if the outer table has been sorted.

In addition, the optimizer may also choose to perform the sort before the join, if it expects that the join will make a later sort more expensive. A later sort could be required to support a GROUP BY, DISTINCT, ORDER BY or merge join.

Merge Join

Merge join (sometimes known as merge scan join or sort merge join) requires a predicate of the form table1.column = table2.column. This is called an *equality join predicate*. Merge join requires ordered input on the joining columns, either through index access or by sorting. In order for a merge join to be used, the join column cannot be a LONG field column or a large object (LOB) column.

The joined tables are scanned simultaneously. The outer table of the merge join is scanned just once. The inner table is also scanned once unless there are repeated values in the outer table. If there are repeated values in the outer table, a group of rows in the inner table may be scanned again. For example, if column A in tables T1 and T2 has the following values:

```
Outer Table T1: column A      Inner Table T2: column A
------------------------      ------------------------
          2                             1
          3                             2
          3                             2
                                        3
                                        3
```

the steps for doing the merge join are:

- Read the first row from T1. The value for A is “2”
- Scan T2 until a match is found, and then join the two rows
- Keep scanning T2 while the columns match, joining rows.
- When the “3” in T2 is read, go back to T1 and read the next row
- The next value in T1 is “3,” which matches T2, so join the rows
- Keep scanning T2 while the columns match, joining rows
- The end of T2 is reached
- Go back to T1 to get the next row — note that the next value in T1 is the same as the previous value from T1, so T2 is scanned again starting at the first “3” in T2 (the database manager remembers this position).

Outer versus Inner Determination

When joining, how are the inner and outer tables determined? The following are general guidelines for how the optimizer decides which table will be the inner and which will be the outer.

The order in which the tables are accessed is particularly important for a **nested loop join** because the outer table is accessed once but the inner table is accessed once for each row of the outer table. The optimizer chooses the outer and inner tables based on cost estimates. These cost estimates are influenced by the following factors:

- Size

  The smaller table is often chosen to be the outer table to reduce the number of times the inner table must be re-accessed. However, prefetch can cause just the opposite to be true. Prefetching can reduce the cost of accessing a large table substantially. However, usually prefetching is only effective for the outer table of a

join. Therefore, the larger table may be accessed first. See “Prefetching Data into the Buffer Pool” on page 405 for more information.

- Predicates

  A table is more likely to be chosen as the outer table if selective predicates can be applied to it because the inner table is only accessed for rows which satisfy the predicates applied to the outer table.

- Buffering

  If the entire inner table must be scanned for each row of the outer table (that is, an index lookup cannot be performed on the inner table), the smaller of the two tables may be chosen as the inner table to take advantage of buffering. This will be influenced by table size and buffer pool size. Note that since join decisions are influenced by buffer pool size, the access plan for your applications may change, if you rebind your applications to the database, after changing the buffer pool size.

  Your ability to create more than one buffer pool, and change the size of that buffer pool, and control the table spaces that use that buffer pool, can affect when buffering is used within inner and outer tables.

- Indexes

  If it is possible to do an index lookup on one of the tables, then that table is a good candidate to use as the inner table. It could then be accessed with an index key lookup using the outer table's join key predicate as one of the key values. If a table does not have an index, it would not be a good candidate for the inner table since in that case the entire inner table would have to be scanned for every row of the outer table.

- Order requirements

  The table associated with a required order might be assessed first. For example, if the output of the join between t1 and t2 was to be ordered on t1.c, accessing t1 as the outer with an index on t1.c might be a good choice. The output of the join would be ordered and no sort would be required.

  ```
  SELECT * FROM t1, t2
    WHERE t1.a = t2.b
    ORDER BY t1.c
  ```

The order in which the tables are accessed is somewhat less important for a **merge join** because both the inner and outer tables are read only once. However, portions of the inner table which correspond to duplicate join values in the outer are kept in an in-memory buffer. The buffer is reread if the next outer row is the same as the previous outer row, otherwise the buffer is reset. If the number of duplicate join values exceeds the capacity of the in-memory buffer, not all of the duplicates are kept. This will only happen when the duplication on any value is large and the value has a matching value in the outer table.

With all of these considerations for duplicate values, in most cases it is the table with fewer duplicates that will be chosen as the outer table in a join. Ultimately, however, the optimizer chooses the outer and inner tables based on detailed cost estimates.

Search Strategies for Selecting Optimal Join

The optimizer can determine optimal join methods using different search strategies. The search strategy that will be used is determined by the optimization class in use (see “Adjusting the Optimization Class” on page 283). The search strategies and their characteristics are:

- Greedy join enumeration
  - Efficient with respect to space and time
  - Single direction enumeration; that is, once a join method is selected for two tables, it will not be changed during further optimization
  - May miss best access plan when joining many tables. If your query only joins two or three tables, the access plan chosen by the greedy join enumeration will be the same as the access plan chosen by dynamic programming join enumeration. This is particularly true if the query has many join predicates (either explicitly specified, or implicitly generated through predicate transitive closure) on the same column.
- Dynamic programming join enumeration
  - Space and time requirements grow exponentially larger as the number of tables being joined increases
  - Efficient and exhaustive search for best access plan
  - Similar to strategy used by DB2 for MVS/ESA.

The join enumeration algorithm is a key determinant of the number of plan combinations that are explored by the optimizer.

Search Strategies for Star Join

In general, the tables referenced in a query should be connected by join predicates. If two tables are joined without the presence of a join predicate, the Cartesian product of the two tables is formed. That is, every qualifying row of the first table is joined with every qualifying row of the second, creating a result table consisting of the cross product of the size of the two tables that is typically very large. Since such a plan is unlikely to perform very well, the optimizer avoids even determining the cost of such an access plan. The only exception to this occurs when the optimization class is set to 9, or the following special case for “Star Schemas.” For more information, see “Adjusting the Optimization Class” on page 283.

The cases where access plans involving Cartesian products perform well are usually large decision support databases designed with the Star Schema technique. The star schema is a database design in which the bulk of the raw data is kept in a single large table with many columns and is commonly known as a “fact” table. Many of the columns contain encoded values that characterize the dimensions of the particular datum stored in the fact table. In order to allow easy analysis of some subset of the facts, dimension tables are used to decode the encoded values. A typical query would consist of multiple local predicates referencing decoded values in the dimension tables and would contain join predicates connecting the dimension tables to the fact table. For these kinds of queries it may be beneficial to perform the Cartesian product of multiple

small dimension tables before accessing the large fact table. This technique is beneficial when multiple join predicates match a multi-column index.

DB2 has the ability to recognize queries against databases designed with star schemas having at least three (3) dimension tables, and to increase the search space to include potential plans that involve forming the Cartesian product of dimension tables. If the plan involving the Cartesian products has the lowest estimated cost, it will be selected by the optimizer.

The Star Schema technique discussed above was focussed on the situation where primary key indexes were used in the join. Another scenario could involve foreign key indexes. Given that the foreign key columns in the fact table are single-column indexes and that there is a relatively high selectivity across all dimension tables, the following Star Join technique can be used:

1. Each dimension table is processed by:
   - Performing a semi-join between the dimension table and the foreign key index on the fact table
   - Hashing the row ID (RID) values to dynamically create a bitmap.
2. Each bitmap is used with “and” predicates against the previous bitmap (see “Multiple Index Access” on page 354).
3. Determine the surviving RIDs after processing the last bitmap.
4. Optionally sort these RIDs.
5. Fetch a base table row.
6. Re-join the fact table with each of its dimension tables, accessing the dimension tables' columns that are needed for the SELECT clause
7. Reapply the predicates (residual predicates)

Using this technique, there is no requirement to have multi-column indexes.

Composite Tables

Another important parameter determines the shape of the sequence of joins in a query. The result of joining a pair of tables is a new table known as a composite. Typically, this resulting composite table becomes the outer table of a join with another inner table. This is known as a “composite outer.” In some situations, particularly when using the greedy join enumeration technique, it is useful to take the result of joining two tables and make that the inner table of a later join. When the inner table of a join itself consists of the result of joining two or more tables, we say that the plan contains a “composite inner.” For example, in the following query:

```
SELECT COUNT(*)
FROM T1, T2, T3, T4
WHERE T1.A = T2.A AND
      T3.A = T4.A AND
      T2.Z = T3.Z
```

it may be beneficial to join table T1 and T2 (T1xT2), then join T3 to T4 (T3xT4) and finally select the first join result as the outer and the second join result as the inner. In the final plan ((T1xT2) x (T3xT4)) the join result (T3xT4) is known as a composite inner. Depending on the query optimization class, the optimizer places different

constraints on the maximum number of tables that may be the inner table of a join. Composite inners are allowed with optimization classes 5, 7, and 9.

Join Strategies in a Partitioned Database

The following sections describe the join strategies that are possible in a partitioned database environment. The DB2 optimizer automatically selects the best join strategy depending on the requirements of each application. The join strategies are presented here to help you understand what is happening in each strategy. A “table queue” is a mechanism for transferring rows between database partitions, or between processors in a single partition database.

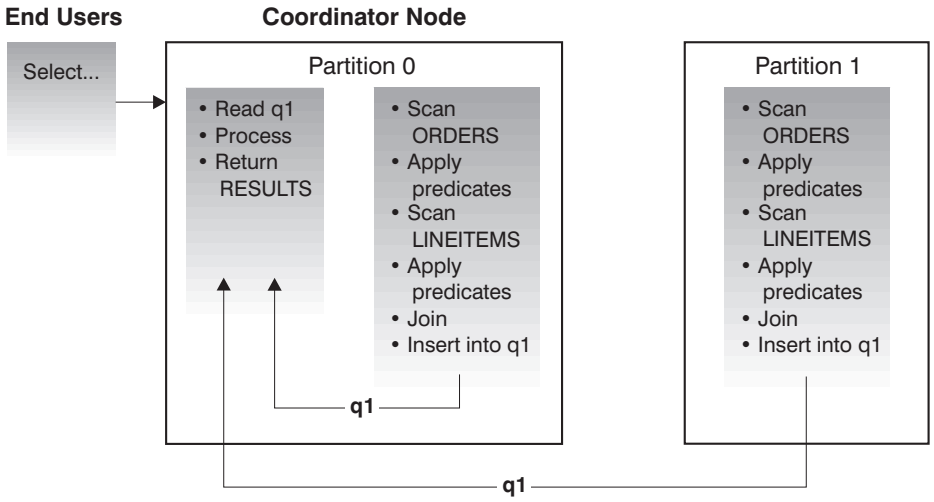
In the descriptions that follow, a *directed* table queue is one whose rows are hashed to one of the receiving database partitions. A *broadcast* table queue is one whose rows are sent to all of the receiving database partitions (that is, it is not hashed). In the diagrams for this section q1, q2, and q3 refer to table queues in the examples. Also the tables that are referenced are divided across two database partitions for the purpose of these scenarios. The arrows indicate the direction in which the table queues are sent. The coordinator node is partition 0.

One consideration for those tables involved in frequent joins in a partitioned database is that of table collocation. Table collocation provides the means in a partitioned database to locate data from one table with the data from another table at the same partition based on the same partitioning key. Once collocated, data to be joined can participate in a query without having to be moved to another database partition as part of the query activity. Only the answer set for the join is moved to the coordinator node. See “Table Collocation” on page 37 for more information on this subject.

For information on join dependencies, refer to the *SQL Reference* manual.

Collocated Joins

For the optimizer to consider a collocated join, the joined tables must be collocated, and all pairs of the corresponding partitioning key must participate in the equijoin predicates. An example is shown in Figure 41 on page 366.

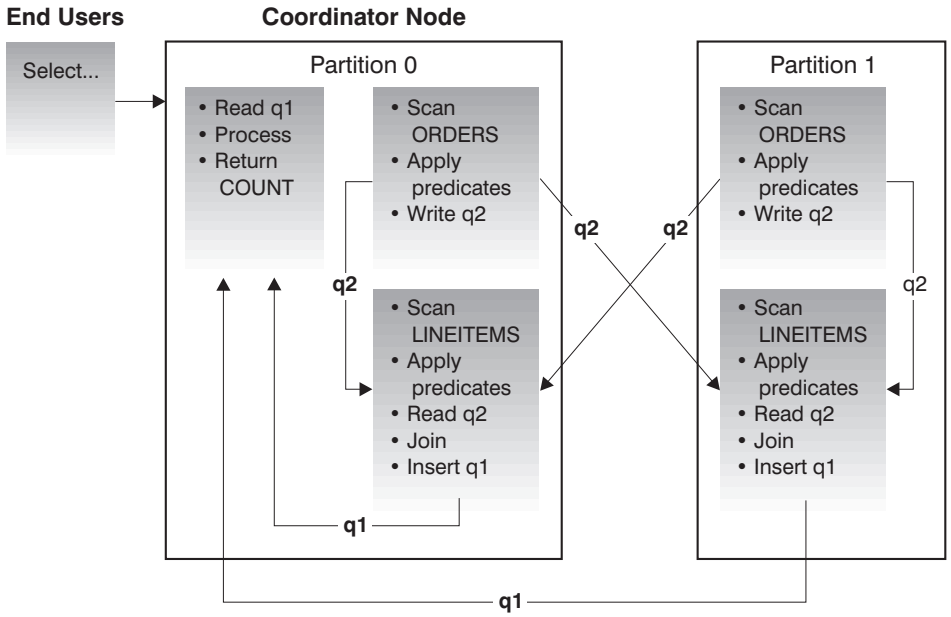


Both the LINEITEM and ORDERS tables are partitioned on the ORDERKEY column. The join is done locally at each database partition. In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 41. Collocated Join Example

Broadcast Outer-Table Joins

This parallel join strategy can be used if there are no equijoin predicates between the joined tables. It can also be used in other situations in which it is the most cost-effective join method. Typically, this would occur when there is one very large table and one very small table, neither of which is partitioned on the join predicate columns. Rather than partition both tables, it may be "cheaper" to broadcast the smaller table to the larger table. An example is shown in Figure 42 on page 367.

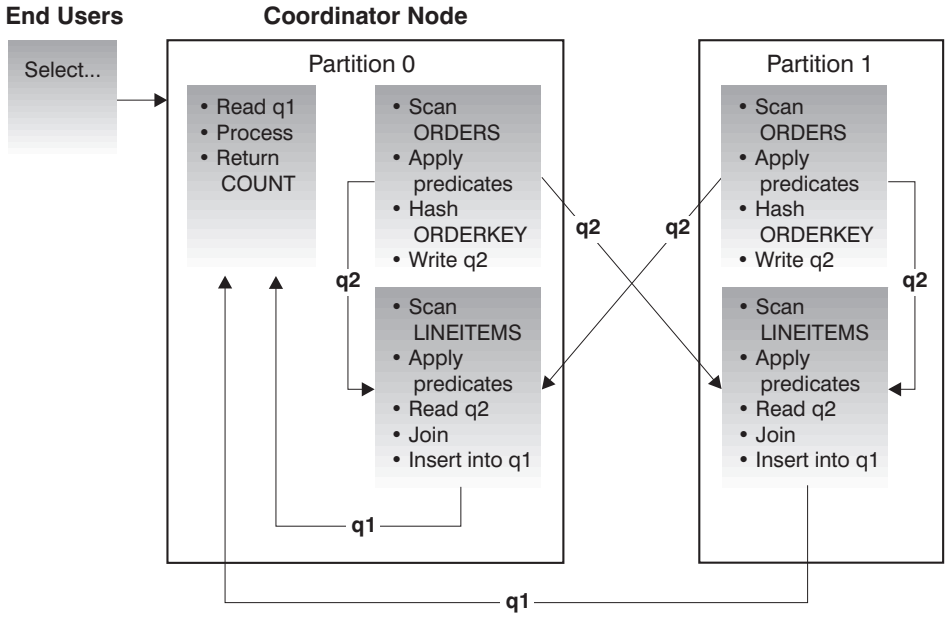


The ORDERS table is sent to all database partitions that have the LINEITEM table. Table queue q2 is broadcast to all database partitions of the inner table.

Figure 42. Broadcast Outer-Table Join Example

Directed Outer-Table Joins

In this join strategy, each row of the outer table is sent to one database partition of the inner table (based on the partitioning attributes of the inner table). The join occurs on this database partition. An example is shown in Figure 43 on page 368.

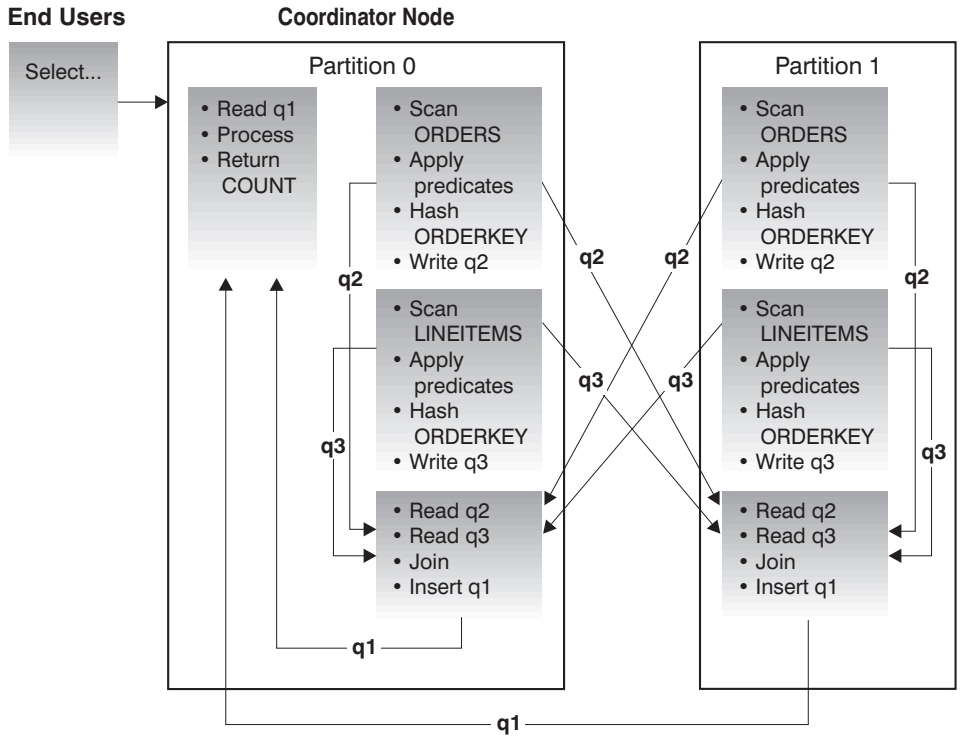


The LINEITEM table is partitioned on the ORDERKEY column.
The ORDERS table is partitioned on a different column.
The ORDERS table is hashed and sent to the correct LINEITEM table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEMS.ORDERKEY.

Figure 43. Directed Outer-Table Join Example

Directed Inner-Table and Outer-Table Joins

With this strategy, rows of the outer and inner tables are directed to a set of database partitions, based on the values of the joining columns. The join occurs on these database partitions. An example is shown in Figure 44 on page 369.

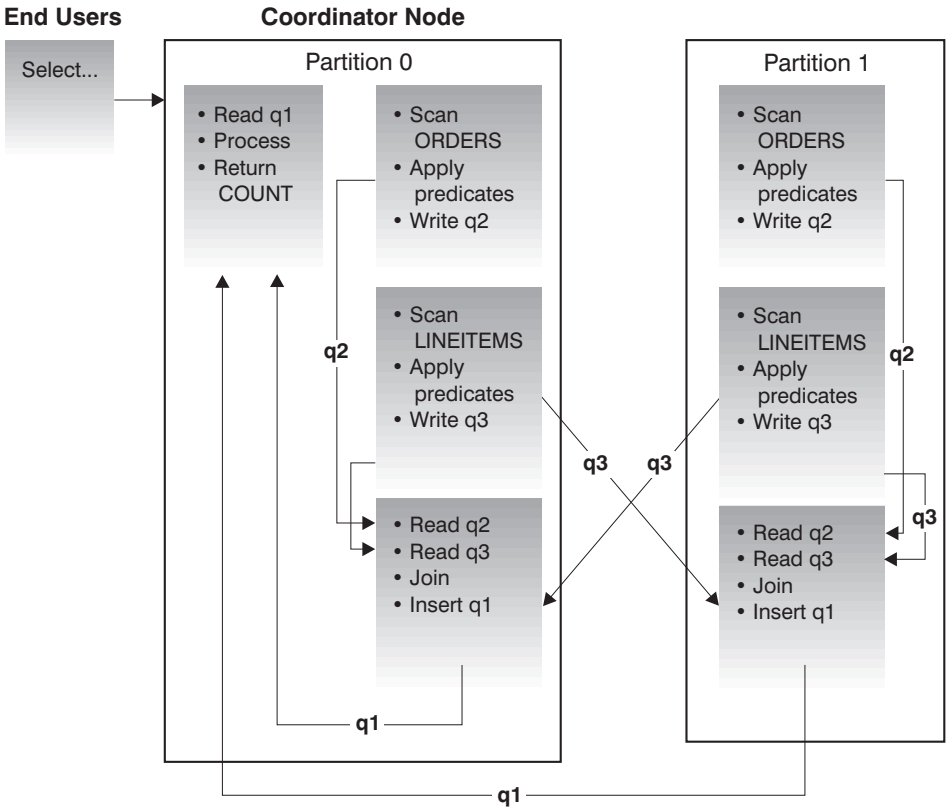


Neither table is partitioned on the ORDERKEY column.
Both tables are hashed and are sent to new database partitions where they are joined.
Both table queue q2 and q3 are directed.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEMS.ORDERKEY

Figure 44. Directed Inner-Table and Outer-Table Join Example

Broadcast Inner-Table Joins

With this strategy, the inner table is broadcast to all the database partitions of the outer join table. An example is shown in Figure 45 on page 370.

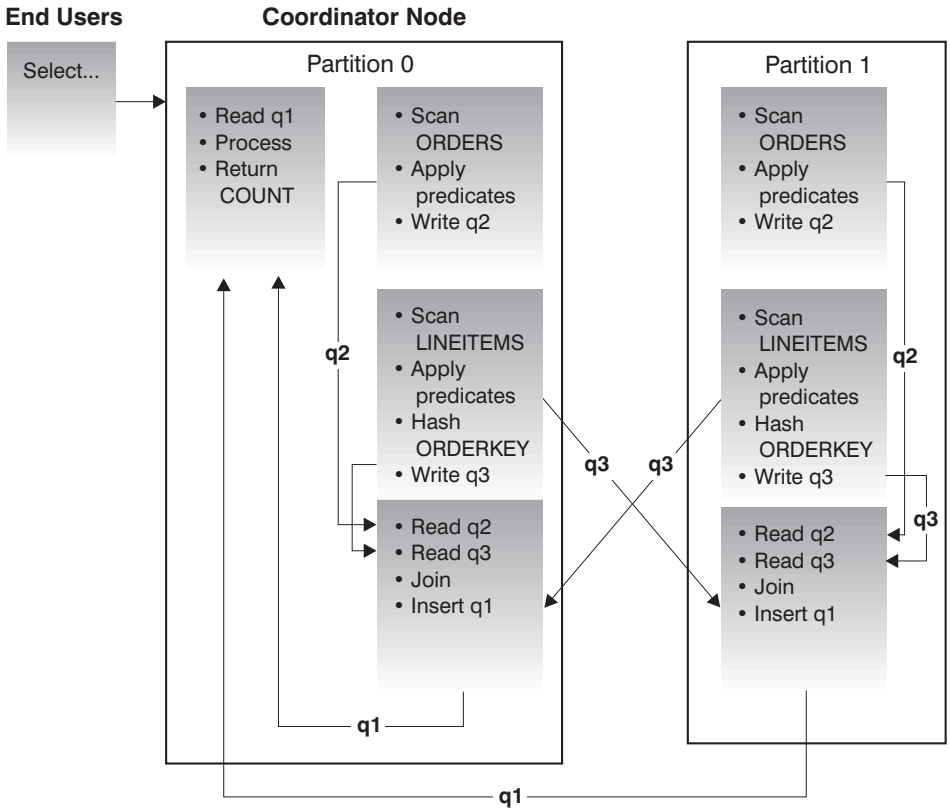


The ORDERS table is sent to all database partitions that have the LINEITEM table. Table queue q2 is broadcast to all database partitions of the inner table.

Figure 45. Broadcast Inner-Table Join Example

Directed Inner-Table Joins

With this strategy, each row of the inner table is sent to one database partition of the outer join table (based on the partitioning attributes of the outer table). The join occurs on this database partition. An example is shown in Figure 46 on page 371.



The ORDERS table is partitioned on the ORDERKEY column.
The LINEITEM table is partitioned on a different column.
The LINEITEM table is hashed and sent to the correct ORDERS table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEMS.ORDERKEY.

Figure 46. Directed Inner-Table Join Example

Table Queues

A table queue is used:

- To pass table data from one database partition to another when using inter-partition parallelism
- To pass table data within a database partition when using intra-partition parallelism
- To pass table data within a database partition when using a single partition database.

Each table queue is used to pass the data in a single direction.

The compiler decides where table queues are required, and includes them in the plan. When the plan is executed, the connections between the database partitions initiate the table queues. The table queues close as processes end.

There are several types of table queues:

- *Asynchronous table queues.* These table queues are known as asynchronous because they read rows in advance of any FETCH being issued by the application. When the FETCH is issued, the row is retrieved from the table queue.

  Asynchronous table queues are used when you specify the FOR FETCH ONLY clause on the SELECT statement. If you are only fetching rows, the asynchronous table queue is faster.

- *Synchronous table queues.* These table queues are known as synchronous because they read one row for each FETCH that is issued by the application. At each database partition, the cursor is positioned on the next row to be read from that database partition.

  Synchronous table queues are used when you do not specify the FOR FETCH ONLY clause on the SELECT statement. In a partitioned database environment, if you are updating rows, the database manager will use the synchronous table queues.

- *Merging table queues.* These table queues preserve order.

- *Non-merging table queues.* These table queues are also known as “regular” table queues. They do not preserve order.

- *Listener table queues.* These table queues are use with correlated subqueries. Correlation values are passed down to the subquery and the results are passed back up to the parent query block using this type of table queue.

Influence of Sorting on the Optimizer

When the optimizer chooses an access plan, it considers the performance impact of sorting data. Sorting occurs when no index exists to satisfy the requested ordering of fetched rows. Sorting could also occur when the sort is determined by the optimizer to be less expensive than an index scan. The optimizer may carry out one of the following actions when sorting the data:

- “Piping” the results of the sort when the query is executed. See “Piped versus Non-Piped Sorts” and “Configuration Parameters Affecting Query Optimization” on page 301.

- Iinternal handling of the sort within the database manager. See “Aggregation and Sort Push-down Operators” on page 373.

Piped versus Non-Piped Sorts

At the completion of a sort, if the final sorted list of data can be read in a single sequential pass, the results can be *piped*. Piping is quicker than the use of other (non-piped) means of communicating the results of the sort. The optimizer chooses to pipe the results of a sort whenever possible.

Independent of whether a sort is piped, the time to sort will depend on a number of factors, including the number of rows to be sorted, the key size and the row width. If the rows to be sorted occupy more than the space available in the sort heap, several sort passes are performed, where each pass sorts a subset of the entire set of rows. Each sort pass is stored in a temporary table in the buffer pool. (As part of the buffer pool management, it is possible that pages from this temporary table may be written to disk.) Once all the sort passes are complete, these sorted subsets must be merged into a single sorted set of rows. If the sort is piped, as the rows are merged they are handed directly to Relational Data Services.

For more information, see “Looking for Indicators of Sorting Performance Problems” on page 413, or the discussion of the *sortheap* configuration parameter in “Configuration Parameters Affecting Query Optimization” on page 301.

Aggregation and Sort Push-down Operators

In some cases, the optimizer can choose to “push-down” a sort or aggregation operation to the Data Management Services component from the Relational Data Services component. Pushing down these operations improves performance by allowing the Data Management Services component to pass data directly to a sort or aggregation routine. Without this push-down, Data Management Services would first pass this data to Relational Data Services, which would then interface with the sort or aggregation routines. For example, the following query benefits from this optimization:

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_DEPT_SALARY
  FROM EMPLOYEE
  GROUP BY WORKDEPT
```

Optimization Strategies for Intra-partition Parallelism

The optimizer may choose an access plan so that a query is executed in parallel within a database partition if a degree of parallelism is specified when the SQL statement is compiled.

At execution time, multiple database agents called “subagents” are created to execute the query. The number of subagents is less than or equal to the degree of parallelism determined when the SQL statement was compiled. For more information on setting the degree of parallelism for SQL statements refer to “Parallel Processing of Applications” on page 298. For more information on agents and subagents, refer to “Database Agents” on page 417.

In a partitioned database, the degree of parallelism applies to each partition. For example, the portion of the query that is executing at a given database partition is further parallelized based on the degree of parallelism determined at that database partition for that SQL statement.

The access plan is parallelized by dividing it into a portion that is run by each subagent and a portion that is run by the coordinating agent. The subagents pass data through table queues to the coordinating agent or to other subagents. In a partitioned database,

subagents may send or receive data through table queues from subagents in other database partitions.

This section describes parallelization strategies within a single database partition.

Parallel Scan Strategies

Relational scans and index scans can be performed in parallel on the same table or index. For parallel relational scans, the table is divided into ranges of pages or rows. A range of pages or rows is assigned to a subagent. A subagent scans its assigned range and is assigned another range when it has completed its work on the current range.

For parallel index scans, the index is divided into ranges of records based on index key values and the number of index entries for a key value. The parallel index scan proceeds like the parallel table scan with subagents being assigned a range of records. A subagent is assigned a new range when it has complete its work on the current range.

The scan unit (either a page or a row) and the scan granularity are determined by the optimizer.

The parallel scan provides an even distribution of work among the subagents. The goal of the parallel scan is to balance the load among the subagents and keep them equally busy. If the number of busy subagents equals the number of available processors and the disks are not overworked with I/O requests, then the machine resources are being used effectively.

Other access plan operations may cause data imbalance as the query executes. The optimizer chooses parallel strategies so that data balance is maintained.

Parallel Sort Strategies

The optimizer may choose one of the following parallel sort strategies:

Round-robin Sort

This is also known as a “redistribution sort.” This is an efficient shared memory sort that attempts to redistribute the data as evenly as possible to all subagents. It uses a round-robin clock type algorithm to provide the even distribution. It first creates an individual sort for each subagent. During the insert phase, subagents insert into each of the individual sorts in a round-robin fashion. This achieves a more even distribution of data.

Partitioned Sort

This is similar to the round-robin sort in that a sort is created for each subagent. The subagents apply a hash function to the sort columns to determine into which sort a row should be inserted. For example, if the inner and outer of a merge join are a partitioned sort, a subagent can use merge join to join the corresponding partitions. This allows the merge join to execute in parallel.

Replicated Sort

This sort is used where all subagents require all the sort output. One sort is created and subagents are synchronized during insertion into the sort. When the sort is completed, each subagent reads the entire sort. This sort may be used to rebalance the data stream if the number of rows is small.

Shared Sort

This sort is the same as a replicated sort, except the subagents open a parallel scan on the sorted result. This distributes the data among the subagents in a way similar to the round-robin sort.

Parallel Temporary Tables

Subagents can cooperate to produce a temporary table by inserting rows into the same table. This is called a `shared temporary table`. The subagents can open private scans or parallel scans on the shared temporary table depending on whether the data stream is to be replicated or partitioned.

Parallel Join Strategies

Join operations can be performed in parallel by subagents. Parallel join strategies are determined by the characteristics of the data stream.

A join can be parallelized by partitioning and/or replicating the data stream on the inner and outer of the join. For example, a nested loop join can be parallelized if its outer stream is partitioned due to a parallel scan and the inner stream is reevaluated independently by each subagent. A merged join can be parallelized if its inner and outer streams are value-partitioned due to partitioned sorts.

Chapter 13. SQL Explain Facility

The SQL explain facility is part of the SQL Compiler that can be used to capture information about the environment where the static or dynamic SQL statement is compiled. The information captured allows you to understand the structure and potential execution performance of SQL statements, including:

- Sequence of operations to process the query
- Cost information
- Predicates and selectivity estimates
- Statistics for all objects referenced in the SQL statement at the time of the explain.

This information can help you:

- Understand the execution plan chosen for a query
- Assist in designing application programs
- Determine when an application should be rebound
- Assist in database design.

The following topics are provided:

The explain output is stored in relational tables and, as an option, in a format which may be graphically displayed using the Visual Explain tool. You should consider using the explain tables to find those queries that are of interest to you. For more information on the tables used by the explain facility and how to create those tables, see Appendix K, " Explain Tables and Definitions" on page 763.

Choosing an Explain Tool

DB2 provides the most comprehensive explain facility in the industry with detailed optimizer information on the access plan chosen for an explained SQL statement. Several methods are provided to give you the flexibility you need to capture and access explain information.

Detailed optimizer information that allows for in-depth analysis of an access plan is kept in explain tables separate from the actual access plan itself. There are three ways to get information from the explain tables:

1. Write your own queries (based on the explain table descriptions as shown in Appendix K, " Explain Tables and Definitions" on page 763)
2. Use the `db2exfmt` tool
3. Use Visual Explain (to view explain snapshot information)

The explain tables are accessible on all supported platforms and contain information for both static and dynamic SQL statements. You can access the explain tables using SQL statements which allows for easy manipulation of the output and for comparison among different queries, or for comparisons of the same query over time. When using the explain tables, you are required to create your own statements to access the tables. If you wish the information from the explain tables to be presented in a predefined format, you can use the `db2exfmt` tool. For more information on the invocation, syntax, and options of this tool, type `db2exfmt -h` on the command line.

Note: The location of this tool (and others like `db2batch`, `dynexpln`, `db2vexp`, and `db2_all`) is in the *misc* subdirectory of the *sqllib* directory. If this tool has been moved from this path, then the command line entry mentioned above may not work.

Visual Explain allows for the analysis of access plan and optimizer information from the explain tables through a graphical interface. Both static and dynamic SQL statements can be analyzed using this tool. Visual Explain is typically invoked from within the Control Center. The Control Center is available from the command line by typing `db2cc`. Also, Visual Explain can be invoked directly from the command line for a single SQL statement using the `db2vexp` command. On some platforms, Visual Explain can be invoked using a folder from within the DB2 Universal Database folder. Visual Explain is not available on all supported platforms. You should check the *Quick Beginnings* for your platform to see if Visual Explain is supported. Visual Explain does allow you to view snapshots captured or taken on another platform. For example, a Windows NT Client can graph snapshots generated on a DB2 for HP-UX server. To do this, both of the platforms must be at a Version 5 level or later. The output from Visual Explain is not easily manipulated for further analysis nor is the information accessible to other applications. For more information on the `db2vexp` command, type `db2vexp -h` on the command line or see the *Command Reference* manual. For more information on Visual Explain in general, refer to the online help in the `Control Center` by typing `db2cc`.

Information about access plans for static SQL statements is generated and stored in the system catalog as part of a package. To see the access plan information available for one or more packages, the `db2expln` tool is available from the command line. `db2expln` shows the actual implementation of the chosen access plan. It does not show optimizer information.

The `dynexpln` tool, which uses `db2expln` within it, provides a quick way to explain dynamic SQL statements that contain no parameter markers. This use of `db2expln` from within `dynexpln` is done by transforming the input SQL statement into a static statement within a pseudo-package. When this occurs, the information may not always be completely accurate. If complete accuracy is desired, you should use the Explain facility.

The `db2expln` tool does provide a relatively compact and English-like overview of what operations will occur at run-time by examining the actual access plan generated (see page 341 for more information on how the code is generated). Additional details on using `db2expln` and interpreting the output can be found in Appendix L, “SQL Explain Tools (db2expln and dynexpln)” on page 783.

Table 35 on page 379 summarizes the different tools available with the DB2 explain facility and their individual characteristics. Use this table to select the tool most suitable for your environment and needs.

Table 35. Explain Facility Tools

Desired Characteristics	Visual Explain	db2vexp	Explain tables	db2exfmt	db2expln	dynexpln
GUI-interface	Yes	Yes				
Text output				Yes	Yes	Yes
"Quick and dirty" static SQL analysis					Yes	
Static SQL supported	Yes		Yes	Yes	Yes	
Dynamic SQL supported	Yes	Yes	Yes	Yes		Yes*
CLI applications supported	Yes		Yes	Yes		
Available to DRDA Application Requesters			Yes			
Detailed optimizer information	Yes	Yes	Yes	Yes		
Suited for analysis of multiple statements			Yes	Yes	Yes	Yes
Information accessible from within an application			Yes			

Notes on this table:

***** Indirectly using db2expln; there are some limitations.

Using the SQL Explain Facility

The different means of capturing explain information include using:

1. EXPLAIN and EXPLSNAP BIND/PREP options
2. CURRENT EXPLAIN MODE and CURRENT EXPLAIN SNAPSHOT special registers
3. EXPLAIN SQL statement
4. `db2vexp` tool (also directly calls Visual Explain to display the information)

There are three reasons you may wish to collect and use explain data:

1. To understand the steps (the access plan) that the database manager must perform to satisfy your query. "Data Access Concepts and Optimization" on page 349 provides information which you may need to reference if you wish to understand the explain output.
2. To help evaluate your performance tuning initiatives. There are a number of actions you can take to help improve the performance of your queries. Many of these possible actions are described in sub-topics of the following:
   - Chapter 9, "Application Considerations" on page 265
   - Chapter 10, "Environmental Considerations" on page 301

- Chapter 11, “System Catalog Statistics” on page 311.

After making a change in any of these areas, you can use the SQL explain facility to determine the impact, if any, that the change has on the access plan chosen. For example, if you add an index based on the recommendations provided in “Index Management” on page 305, the explain data can help you determine whether the index is, in fact, being used as you expected.

While the explain output will provide you with information to allow you to determine the access plan that was chosen and its relative cost, the only way to accurately measure the performance improvement for a query is to use benchmark testing techniques, as described in Chapter 18, “Benchmark Testing” on page 447.

3. To help you understand the reasons for changes in query performance, you need to have the explain information both before and after your change in order to analyze the impact. Therefore, when compiling a SQL statement to the database, you should:
   - Use the explain facility to capture the plan information before your changes, and save the resulting explain tables.
   - Save and/or print the current catalog statistics if you do not want to, or cannot, access Visual Explain to view this information. (The `db2look` productivity tool, described in “Modelling Production Databases” on page 337, could be used to help perform this task.)
   - Save and/or print the data definition language (DDL) statements, including those for CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE TABLESPACE.

   The above information provides you with a “before” picture that you can use as a reference point for future analysis. For dynamic SQL statements, you can also collect this information when you run your application for the first time. For static SQL statements, you can also collect this information at bind time.

   When you wish to analyze the reason for a performance change, you can compare the “before” data to information you collect about the query and environment when you are starting your analysis (the “after” data).

   As a simple example, your analysis could show that an index is no longer being used as part of the access path. Using the catalog statistics information in Visual Explain, you might notice that the number of index levels (NLEVELS column) is now substantially higher than when the query was first bound to the database. You might then choose to:
   - Reorganize the index
   - Collect new statistics for your table and indexes
   - Gather explain information when rebinding your query.

   Following these actions, you might notice that the index is once again being used in the access plan and that performance of the query is no longer a problem.

Introductory Concepts for Explain

You can use explain information to analyze the access plan that the optimizer has chosen based on the choices described in “Data Access Concepts and Optimization” on page 349. For example, explain information may indicate that an index scan (see

"Index Scan Concepts" on page 349) was chosen by the optimizer. In addition, it can also allow you to determine the following:

- How many index columns are used as search criteria, as described in "Range Delimiting and Index SARGable Predicates" on page 357
- Whether index-only access is used, as described in "Index-Only Access" on page 353
- Whether list prefetch will be used to read the pages, as described in "Understanding List Prefetching" on page 407.

As another example, the explain information could also help you understand how two tables are joined:

- The join method
- The order in which the tables are joined
- The occurrence and type of sorts.

Although you can use explain for SELECT, SELECT INTO, UPDATE, INSERT, VALUES, VALUES INTO, and DELETE SQL statements, the primary use of explain is to observe the access paths for the SELECT parts of your statements.

To satisfy an SQL query, the database manager typically:

- Uses one or more data objects (a table, an index, or both)
- Performs one or more operations (for example, table scan, index scan, and join)
- Returns the result set to the calling application.

For a simple SQL query, such as:

```
SELECT DEPTNO, DEPTNAME
  FROM DEPARTMENT
```

the following, graphical representation of the steps performed could be displayed by Visual Explain:

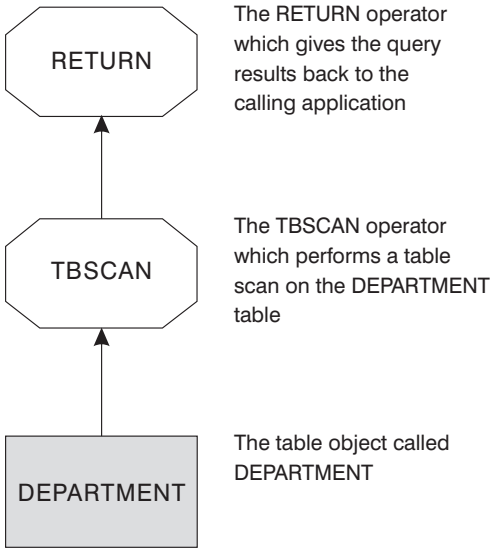


Figure 47. Graphical Display of Explain Output

The following topics discuss the type of details you can view for objects and operators:

- "Explain Information for Data Objects"
- "Explain Information for Data Operators" on page 383

Explain Information for Data Objects

A single access plan may use one or more data objects to satisfy the SQL statement.

Object Statistics: The explain facility records facts about the object, such as:

- The creation time
- The last time that statistics were collected for the object (see Chapter 11, "System Catalog Statistics" on page 311)
- An indication of whether or not the data in the object is ordered
- The number of columns in the object
- The estimated number of rows in the object
- The number of pages that the object occupies in the buffer pool
- The total estimated overhead, in milliseconds, for a single random I/O to the specified table space where this object is stored
- The estimated transfer rate, in milliseconds, to read a 4K page from the specified table space
- Prefetch and extent sizes, in 4K pages
- The degree of data clustering with the index
- The number of leaf pages used by this object's index and the number of levels in the tree
- The number of distinct full key values in this object's index
- The total number of overflow records in the table.

Explain Information for Data Operators

A single access plan may perform several operations on the data to satisfy the SQL statement and provide results back to you. The SQL compiler determines the operations required; for example, a table scan, an index scan, a nested loop join, or a group-by. Details of many of these operators are provided in “Data Access Concepts and Optimization” on page 349.

In addition to showing the various operators used in an access plan, explain information is also available for each operator as well as the cumulative effects of the access plan.

Estimated Cost Information: The following estimated, cumulative costs can be displayed for the operators. These costs are for the chosen access plan, up to and including the operator for which the information is captured.

- The total cost (in timerons)
- The number of 4KB page I/Os
- The number of CPU instructions
- The cost (in timerons) of fetching the first row, including any initial overhead required
- The communication cost (in frames)

Timerons are a made-up, relative unit of measure.

Operator Properties: The following information is recorded by the explain facility to describe the properties of each operator:

- The set of tables that have been accessed
- The set of columns that have been accessed
- The columns on which the data is ordered, if the optimizer determined that this ordering can be used by subsequent operators
- The set of predicates that have been applied
- The estimated number of rows that will be returned (cardinality).

How Explain Information is Organized

All explain information is organized around the concept of an explain instance. An explain instance represents one invocation of the explain facility for one or more SQL statements. An explain instance represents the explain information for:

- All the eligible SQL statements in one package for **static** SQL statements
- One particular SQL statement for **dynamic** SQL statements
- Each EXPLAIN SQL statement (whether dynamic or static).

The explain information captured within one explain instance includes the SQL Compilation environment as well as the access plan chosen to satisfy the SQL statement being compiled. Explain information is organized into 3 subsets:

Explain Instance Information	Compilation environment information captured for each explain instance.
Explain Snapshot Information	Information used by Visual Explain.

Explain Table Information	Information collected when explain table information is requested.

Explain Instance Information

Explain instance information is stored in the EXPLAIN_INSTANCE table. Additional specific information about each SQL statement explained within an explain instance is stored in the EXPLAIN_STATEMENT table.

Explain Instance Identification: You can uniquely identify each explain instance and correlate the information for the SQL statements to a given invocation of the facility with this information:

- The user who requested the explain information
- When the explain request began
- The name of the package from which the explained SQL statement came
- The schema of the package from which the explained SQL statement came.
- An indication whether a snapshot was part of the explain request.

Environmental Settings: Environmental information concerning how the SQL compiler optimized your queries is captured. The environmental information includes the following:

- The version and release number for the level of DB2 being used.
- The degree of parallelism used to compile the query. The CURRENT DEGREE special register, the DEGREE bind option, the SET RUNTIME DEGREE API, and the *dft_degree* configuration parameter may be used to determine the degree of parallelism to be used when compiling a particular query.
- Whether the SQL statement was dynamic or static.
- The query optimization class used to compile the query. See “Adjusting the Optimization Class” on page 283 for more information.
- The type of cursor blocking specified when compiling the query. For more information about cursors, refer to the *SQL Reference* manual. For more information about cursor blocking, see “Row Blocking” on page 291.
- The isolation level used when compiling the query. See “Concurrency” on page 265 for more information.
- The values of various configuration parameters when the query was compiled. See “Configuration Parameters Affecting Query Optimization” on page 301 for more information about the configuration parameters that can affect query optimization, including the following parameters that are recorded when an explain snapshot is taken:
  - “Buffer Pool Size (buffpage)” on page 470
  - “Sort Heap Size (sortheap)” on page 482
  - “Average Number of Active Applications (avg_appls)” on page 509
  - “Database Heap (dbheap)” on page 472
  - “Maximum Storage for Lock List (locklist)” on page 477
  - “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 500

– “CPU Speed (cpuspeed)” on page 565
– “Communications Bandwidth (comm_bandwidth)” on page 565.

SQL Statement Identification: For each explain instance, multiple SQL statements may have been explained. Along with information that uniquely identifies the explain instance, the following information helps identify each individual SQL statement.

- The type of statement: SELECT, DELETE, INSERT, UPDATE, positioned DELETE, positioned UPDATE.
- The statement and section number of the package issuing the SQL statement, as recorded in SYSCAT.STATEMENTS catalog view.

Within the EXPLAIN_STATEMENT table, the QUERYTAG and QUERYNO fields contain identifiers and are set for you as part of the explain process.

For dynamic explain SQL statements submitted during a CLP or CLI session, when EXPLAIN MODE or EXPLAIN SNAPSHOT is active, the QUERYTAG is set to “CLP” or “CLI.” When this happens, the QUERYNO is defaulted to a number that is incremented by one or more for each statement.

For all other dynamic explain SQL statements (not from CLP, CLI, or using the EXPLAIN SQL statement) the QUERYTAG is set to blanks, and the QUERYNO will always be “1.”

Cost Estimation: For each statement explained, an estimate of the relative cost of executing the chosen access plan is recorded. This cost is given using a made-up, relative unit of measure called *timerons*. Estimates of elapsed times are **not** provided, for the following reasons:

- The SQL optimizer does not estimate elapsed time but rather resource consumption.
- The optimizer does not model all factors that can affect elapsed time; it ignores those that do not affect the efficiency of the access plan. The elapsed time **is** affected by a number of run-time factors including: the system workload; the amount of resource contention; the amount of parallel processing and I/O; the cost of returning rows to the user; and the communication time between the client and server.

Statement Text: For each statement explained, two versions of the text of the SQL statement are recorded. One version is the text as received by the SQL Compiler. The other is a version of the statement text that has been reverse-translated from the internal compiler representation of the query. This translation, while looking similar to other SQL statements, does **not** necessarily follow correct SQL syntax nor does it necessarily reflect the actual content of the internal representation as a whole. This translation is provided simply to allow an understanding of the SQL context from which the SQL optimizer chose the access plan. Comparing the user-written statement text to the internal representation of the SQL statement can help you to understand how the SQL compiler has rewritten your query for better optimization. (See “Query Rewrite by the SQL Compiler” on page 342.) It also shows you other elements in the environment

affecting your statement such as triggers and constraints. Some keywords used by this "optimized" text are:

$Cn	The name of a derived column, where n represents an integer value.
$CONSTRAINT$	The tag used to indicate the name of a constraint added to the original SQL statement during compilation. Seen in conjunction with the $WITH_CONTEXT$ prefix.
$DERIVED.Tn	The name of a derived table, where n represents an integer value.
$INTERNAL_FUNC$	The tag used to indicate the presence of a function used by the SQL Compiler for the explained query but not available for general use.
$INTERNAL_PRED$	The tag used to indicate the presence of a predicate added by the SQL Compiler during compilation of the explained query. Again, such a predicate is not available for general use. An internal predicate is used by the compiler to satisfy additional context added to the original SQL statement as the result of triggers and constraints.
RID	The tag used to identify the Row Identifier (RID) column for a particular row.
$TRIGGER$	The tag used to indicate the name of a trigger added to the original SQL statement during compilation. Seen in conjunction with the $WITH_CONTEXT$ prefix.
$WITH_CONTEXT$(...)	This prefix will appear at the start of the text when additional triggers or constraints have been added into the original SQL statement. Following this prefix will appear a list of the names of any triggers or constraints affecting the compilation and resolution of the SQL statement.

Explain Snapshot Information

When an explain snapshot is requested, additional explain information is recorded describing the access plan selected by the SQL optimizer. This information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT table in the format required by Visual Explain. This format is not usable by other applications.

Additional information on the contents of the explain snapshot information is available from Visual Explain itself and in:

- "Explain Information for Data Objects" on page 382
- "Explain Information for Data Operators" on page 383

Explain Table Information

When explain table information is requested, additional information is recorded describing the access plan selected by the SQL optimizer. This information is stored in the following explain tables:

- EXPLAIN_ARGUMENT. This table represents the unique characteristics for each individual operator, if any.

- EXPLAIN_INSTANCE. This table is the main control table for all Explain information. Each row of data in the Explain tables is explicitly linked to one unique row in this table. Basic information about the source of the SQL statements being explained and environment information is kept in this table.
- EXPLAIN_OBJECT. This table identifies those data objects required by the access plan generated to satisfy the SQL statement.
- EXPLAIN_OPERATOR. This table contains all the operators needed to satisfy the SQL statement by the SQL compiler.
- EXPLAIN_PREDICATE. This table identifies which predicates are applied by a specific operator.
- EXPLAIN_STATEMENT. This table contains the text of the SQL statement as it exists for the different levels of Explain information. The original SQL statement as entered by the user is stored in this table along with the version used (by the optimizer) to choose an access plan to satisfy the SQL statement.
- EXPLAIN_STREAM. This table represents the input and output data streams between individual operators and data objects. The data objects themselves are represented in the EXPLAIN_OBJECT table. The operators involved in a data stream are represented in the EXPLAIN_OPERATOR table.

Each rectangular “object” node of Visual Explain corresponds to a row in the EXPLAIN_OBJECT table. Each octagonal “operator” node of Visual Explain corresponds to a row in the EXPLAIN_OPERATOR table. Each link between operators or operator's objects corresponds to a row of the EXPLAIN_STREAM table.

The explain table information is similar in content to that recorded for an explain snapshot, however, this information is stored in ordinary relational tables which can be accessed using standard SQL statements.

Explain tables, like the Visual Explain access plan graph, are designed to reflect the relationships between operators and data objects within the access plan. The following diagram shows the relationships between these tables.

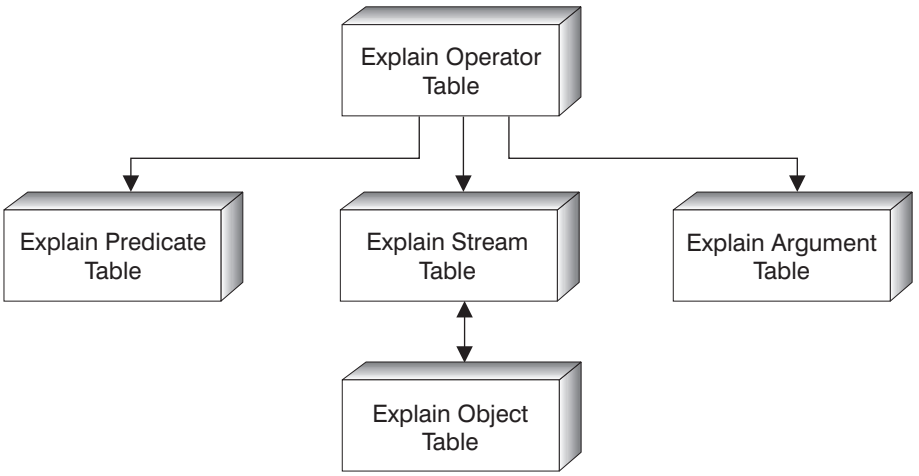


Figure 48. Overview of Explain Table Relationships (not all tables are shown).

It is possible to have explain tables that are common to more than one user. The explain tables can be defined for one user. Aliases can then be defined using the same name for each additional user pointing to the defined tables. Each user sharing the common explain tables must have insert permission on those tables.

See Appendix L, "SQL Explain Tools (db2expln and dynexpln)" on page 783 for more information on the Explain tables and how to create the tables. Additional information on the contents of the explain table information is available in:

- "Explain Information for Data Objects" on page 382
- "Explain Information for Data Operators" on page 383

The `db2exfmt` tool provided in the `misc` subdirectory under the `sqllib` directory can be used to format the contents of the explain tables into a legible, organized output.

Obtaining Explain Data

Before you can obtain explain data for an SQL statement, you must have a set of explain tables defined using the same schema as the authorization ID that invokes the explain facility. See "Table Definitions for Explain Tables" on page 776 for information on how to create the tables.

Capturing Explain Table Information

Once these tables are defined, explain data is captured when an SQL statement is compiled and explain data has been requested:

- For **static** SQL statements, explain table information will be captured when either EXPLAIN ALL or EXPLAIN YES options are specified on the BIND or PREP command; or, a static EXPLAIN SQL statement is used in the source program.

- For **dynamic** SQL statements, explain table information will be captured for any of the following situations:
  - An EXPLAIN SQL statement. All explain information is captured and placed in the explain tables unless the FOR SNAPSHOT clause is used.

    An example of an EXPLAIN SQL statement:

    ```
    EXPLAIN PLAN FOR <any valid DELETE, INSERT, SELECT, SELECT INTO,
       UPDATE, VALUES, or VALUES INTO SQL statement>
    ```

  - The CURRENT EXPLAIN MODE special register is set to YES. This setting will cause the SQL compiler to capture explain data and allow the SQL statement to execute, returning the results of the query.
  - The CURRENT EXPLAIN MODE special register is set to EXPLAIN. This setting will cause the SQL compiler to capture explain data, but will not execute the SQL statement.
  - The EXPLAIN ALL option has been specified on the BIND or PREP command. This setting will cause the SQL compiler to capture explain data for dynamic SQL at run-time, even if the setting of the CURRENT EXPLAIN MODE special register is NO. The SQL statement will also execute, returning the results of the query.

  Note: Explain information is only captured when the SQL statement is compiled. Following the initial compilation, dynamic SQL statements are only recompiled when a change to the environment requires the statement be recompiled. If the same PREPARE statement is issued consecutively for the same SQL statement, the SQL statement will only be compiled, and explain data captured, the first time the PREPARE statement is issued, assuming the environment does not change.

For more information about using the EXPLAIN SQL statement refer to the *SQL Reference* manual. For more information about using the CURRENT EXPLAIN MODE registers, refer to the *SQL Reference* manual. For more information about the BIND and PREP commands, refer to the *Command Reference* manual.

Capturing Explain Snapshot Information

Explain snapshot data is captured when an SQL statement is compiled and explain data has been requested:

- For **static** SQL statements, an explain snapshot will be captured when either EXPLSNAP ALL or EXPLSNAP YES options are specified on the BIND or PREP command; or, a static EXPLAIN SQL statement, using a FOR SNAPSHOT or WITH SNAPSHOT clause, is used in the source program.
- For **dynamic** SQL statements, an explain snapshot will be captured in any of the following situations:
  - An EXPLAIN SQL statement using a FOR SNAPSHOT or a WITH SNAPSHOT clause. The FOR SNAPSHOT clause has no explain table information captured except the information associated with explain snapshot. The WITH SNAPSHOT clause has all explain table information captured in addition to the information associated with explain snapshot.

An example of an explain snapshot using the EXPLAIN SQL statement:

```
EXPLAIN PLAN FOR SNAPSHOT FOR <any valid DELETE, INSERT, SELECT,
   SELECT INTO, UPDATE, VALUES, or VALUES INTO SQL statement>
```

Only an explain snapshot is taken and the captured information is placed in the EXPLAIN_INSTANCE and EXPLAIN_STATEMENT tables.

- The CURRENT EXPLAIN SNAPSHOT special register is set to YES. This setting will cause the SQL compiler to take a snapshot of explain data and allow the SQL statement to execute, returning the results of the query.
- The CURRENT EXPLAIN SNAPSHOT special register is set to EXPLAIN. This setting will cause the SQL compiler to take a snapshot of explain data, but will not execute the SQL statement.
- The EXPLSNAP ALL option has been specified on the BIND or PREP command. This setting will cause the SQL compiler to take a snapshot of explain data at run-time, even if the setting of the CURRENT EXPLAIN SNAPSHOT special register is NO. The SQL statement will also execute, returning the results of the query.

Note: Explain information is only captured when the SQL statement is compiled. Following the initial compilation, dynamic SQL statements are only recompiled when a change to the environment requires the statement be recompiled. If the same PREPARE statement is issued consecutively for the same SQL statement, the SQL statement will only be compiled, and explain data captured, the first time the PREPARE statement is issued, assuming the environment does not change.

For more information about using the EXPLAIN SQL statement and the FOR SNAPSHOT or WITH SNAPSHOT clauses refer to the *SQL Reference* manual. For more information about using the CURRENT EXPLAIN SNAPSHOT registers, refer to the *SQL Reference* manual. For more information about the BIND and PREP commands, refer to the *Command Reference* manual.

Guidelines on Using Explain Output

There are a number of ways in which analyzing the explain data can help you to tune your queries and environment. For example:

- **Are Indexes Being Used?**

  As discussed in “Index Management” on page 305, the proper indexes can have a significant benefit on performance. Using the explain output, you can determine if the indexes you have created to help a specific set of queries are being used. In the explain output, you should look for index usage in the following areas:
  - Join predicates
  - Local predicates
  - GROUP BY clause
  - ORDER BY clause
  - The select list.

You can also use the explain facility to evaluate whether a different index can be used instead of an existing index, or no index at all. After creating a new index, collect statistics for that index (using the RUNSTATS command) and recompile your query. Over time you may notice through the explain data that instead of an index scan, a table scan is now being used. This can result from a change in the clustering of the table data. If the index that was previously being used now has a low cluster ratio, you may want to:

- Reorganize your table to cluster the data according to that index
- Use the RUNSTATS command to update the catalog statistics for the table and index
- Recompile your query
- Re-examine the explain output to determine whether reorganizing your table has impacted the access plan.

- **Is the Type of Access Appropriate for Your Application?**

  You can analyze the explain output and look for types of access to the data that, as a rule, are not optimal for the type of application you are running. For example:

  - **Online Transaction Processing (OLTP) Queries**

    OLTP applications are prime candidates to use index scans with range delimiting predicates, because they tend to return only a few rows that are qualified using an equality predicate against a key column. If your OLTP queries are using a table scan, you may want to analyze the explain data to determine the reasons why an index scan was not used.

  - **Browse-Only Queries**

    The search criteria for a “browse” type query may be very vague, causing a large number of rows to qualify. If the user will usually only look at a few screens of the output data, you may want to try to ensure that the entire answer set need not be computed before some results are returned. In this case, the goals of the user are different from the basic operating principle of the optimizer, which attempts to minimize resource consumption for the entire query, not just the first few screens of data.

    For example, if the explain output shows that both merge scan join and sort operators were used in the access plan, then the entire answer set will be materialized in a temporary table before any rows are returned to the application. In this case, you can attempt to change the access plan by using the OPTIMIZE FOR clause on the SELECT statement. (For more information on the OPTIMIZE FOR clause, see “Quickly Retrieving the First Few Rows Using OPTIMIZE FOR n ROWS” on page 289.) In this way, the optimizer can attempt to choose an access plan that does not produce the entire answer set in a temporary table before returning the first rows to the application.

- **What Type of Join Method is Being Used?**

  If a query joins two tables, you can check the type of join processing being used. Joins involving more rows, such as those in decision-support queries, usually run faster with a merge join. Joins involving only a few rows, such as OLTP queries, typically run faster with nested loop joins. However, there may be extenuating circumstances in either case, such as the use of local predicates or indexes, that

would change how these typical joins would work. (See “Nested Loop Join” on page 359 and “Merge Join” on page 361 for information about how these two join methods operate.)

Visual Explain

Visual Explain can be used to study queries in more detail when compared to the other methods, especially those that contain more complex sequences of operations. Visual Explain is not available on all supported platforms. You should check the *Quick Beginnings* for your platform to see if Visual Explain is supported.

Visual Explain lets you view the access plan for explained SQL statements as a graph. You can use the information available from the graph to tune your SQL queries for better performance. Visual Explain also lets you dynamically explain a SQL statement and view the resulting access plan graph.

The optimizer chooses an access plan and Visual Explain displays the information as an access plan graph in which tables and indexes, and each operation on them, are represented as nodes, and the flow of data is represented by the links between the nodes.

To display an access plan graph, you must have created an explain snapshot. From an access plan graph, you can view the details for:

- Tables and indexes (and their associated columns)
- Operators (such as table scans, sorts, and joins)
- Table spaces and functions.

You can also use Visual Explain to:

- View the statistics that were used at the time of optimization. You can then compare these statistics to the current catalog statistics to help you determine whether rebinding the package might improve performance.
- Determine whether or not an index was used to access a table. If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- View the effects of performing various tuning techniques by comparing the before and after versions of the access plan graph for a query.
- Obtain information about each operation in the access plan, including the total estimated cost and number of rows retrieved (cardinality).

For additional detail on Visual Explain, you should refer to the online information available through the Control Center. The Control Center can be accessed by typing `db2cc` on the command line.

Part 4. Tuning and Configuring Your System

Chapter 14. Operational Performance

The following topics provide information on how you can influence performance of an SQL query during run-time:

- How DB2 Uses Memory
- Managing the Database Buffer Pool
- Managing Multiple Database Buffer Pools
- Prefetching Data into the Buffer Pool
- Configuring I/O Servers for Prefetching and Parallel I/O
- Sorting
- Reorganizing Table Data
- Performance Considerations for DMS Devices
- Managing Initialization Overhead
- Database Agents
- Using the Database System Monitor
- Extending Memory.

The following also provide information on how performance can be influenced:

- Chapter 2, “Designing Your Physical Database” on page 25
- Chapter 9, “Application Considerations” on page 265
- Chapter 10, “Environmental Considerations” on page 301
- Chapter 11, “System Catalog Statistics” on page 311.

How DB2 Uses Memory

Many of the configuration parameters available in DB2 affect memory usage on the system. Some may affect memory on the server, some on the client, and some on both. Furthermore, memory is allocated and de-allocated at different times and from different areas of the system.

A system administrator should also take into consideration balancing overall memory usage on the system. Different applications running on the operating system may use memory in different ways. For example, some applications may use the file system cache, while the database manager uses its own buffer pool for data caching instead of the operating system facility. See “Setting Parameters That Affect Memory Usage” on page 400 for additional considerations.

Figure 49 on page 396 shows that the database manager uses different types of memory.

Memory is allocated for each instance of the database manager at the following times:

- When the database manager is started (`db2start`), the area marked “Global Control Blocks” is allocated, and this area remains allocated until the database manager is stopped (`db2stop`). This area contains information that is needed by the database manager to manage activity across all database connections. When the

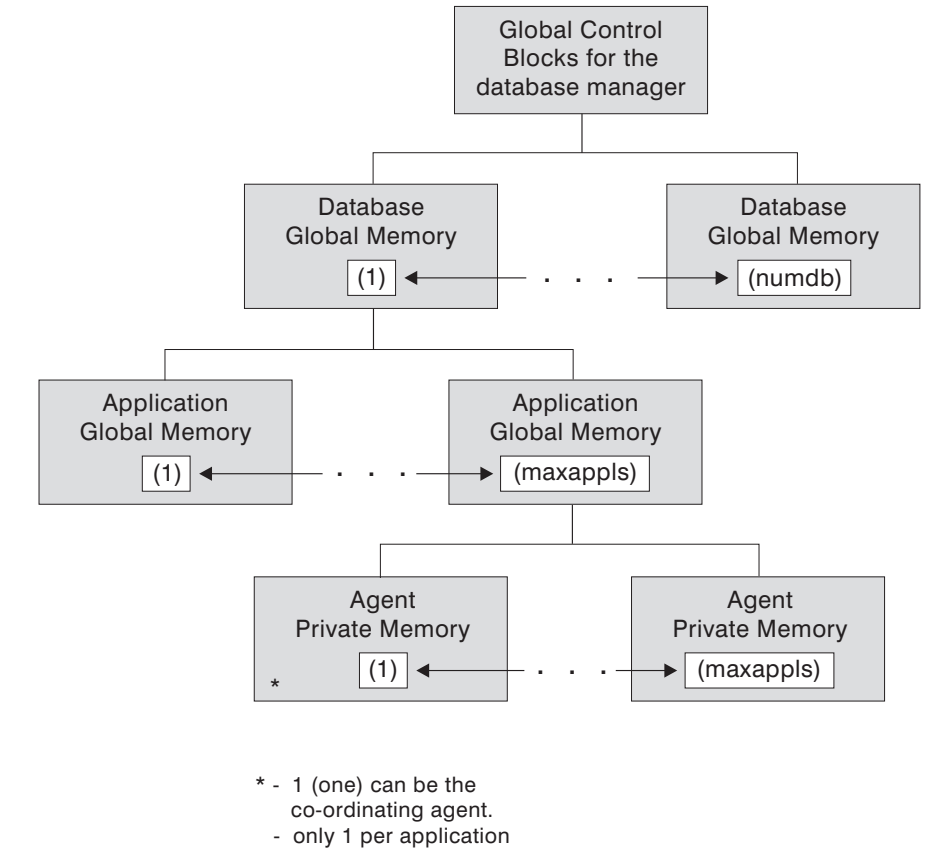


Figure 49. Types of memory used by the database manager

first application connects to a database, both global and private memory areas are allocated.

The “database global memory” is used across all applications that might connect to the database and contains memory areas such as the buffer pools, lock list, database heap and utility heap. The “agent private memory” area is allocated for the agent and contains memory allocations that will be used only by this specific agent, such as the sort heap and the application heap.

For these global and private memory areas, most memory heaps are not allocated until they are actually required. The exception is the package cache, the buffer pool and the lock list, which are allocated for each database partition at the time the database is started.

- In addition to the database global memory and agent/application shared memory, the database manager also allocates “agent private memory” to contain the data that the agent passes to local client applications.

Once a database is already in use by one application, any subsequent connecting applications will only have agent private memory and application control shared memory allocated on their behalf.

For SMP/MPP configurations, there is also an amount of memory required on each database partition for the agents working on behalf of a particular application to share data.

Figure 49 on page 396 shows how configuration parameter settings can affect memory. In particular, the parameters in the following list can limit the amount of memory that is allocated for specific purposes. (In an SMP/MPP database environment, this memory is required on every database partition.)

- *numdb* defines the maximum number of concurrent active databases (in use by different applications). Since each database has its own global memory area, the amount of memory that can potentially be allocated grows if the value of this parameter increases.
- *maxappls* defines the maximum number of applications that can simultaneously connect to a single database. It affects the amount of memory that can potentially be allocated for “Agent Private Memory” and “Application Global Memory” for that database. (Note that this parameter can be set differently for every database.)
- *maxagents* (and *max_coordagents* for SMP/MPP environments) defines the maximum number of database manager agents that can exist simultaneously across all active databases. Along with *maxappls*, these parameters limit the amount of memory allocated for “Agent Private Memory” and “Application Global Memory.” (For information on agents, see “Database Agents” on page 417.)

Figure 50 on page 398 summarizes how much memory must be used to support local applications. The following configuration parameters allow you to control the size of this memory, by limiting the number of "memory segments" (portions of logical memory) and their size.

Database Global Memory

Memory space is required for the database manager to run. This space can be very large, especially in SMP and MPP environments. You can predict and control the size of this space by reviewing the following sections:

- “Database Agents” on page 417. Agents running on behalf of applications require substantial memory space, especially if the value of *maxagents* is not appropriate.
- “FCM Requirements” on page 400. For SMP and MPP systems, Fast Communications Manager (FCM) requires substantial memory space, especially if the value of *fcm_num_buffers* is not appropriate.

Database Global Memory is also affected by the following configuration parameters:

- The number of memory segments is limited by *numdb* (see “Maximum Number of Concurrently Active Databases (numdb)” on page 566).

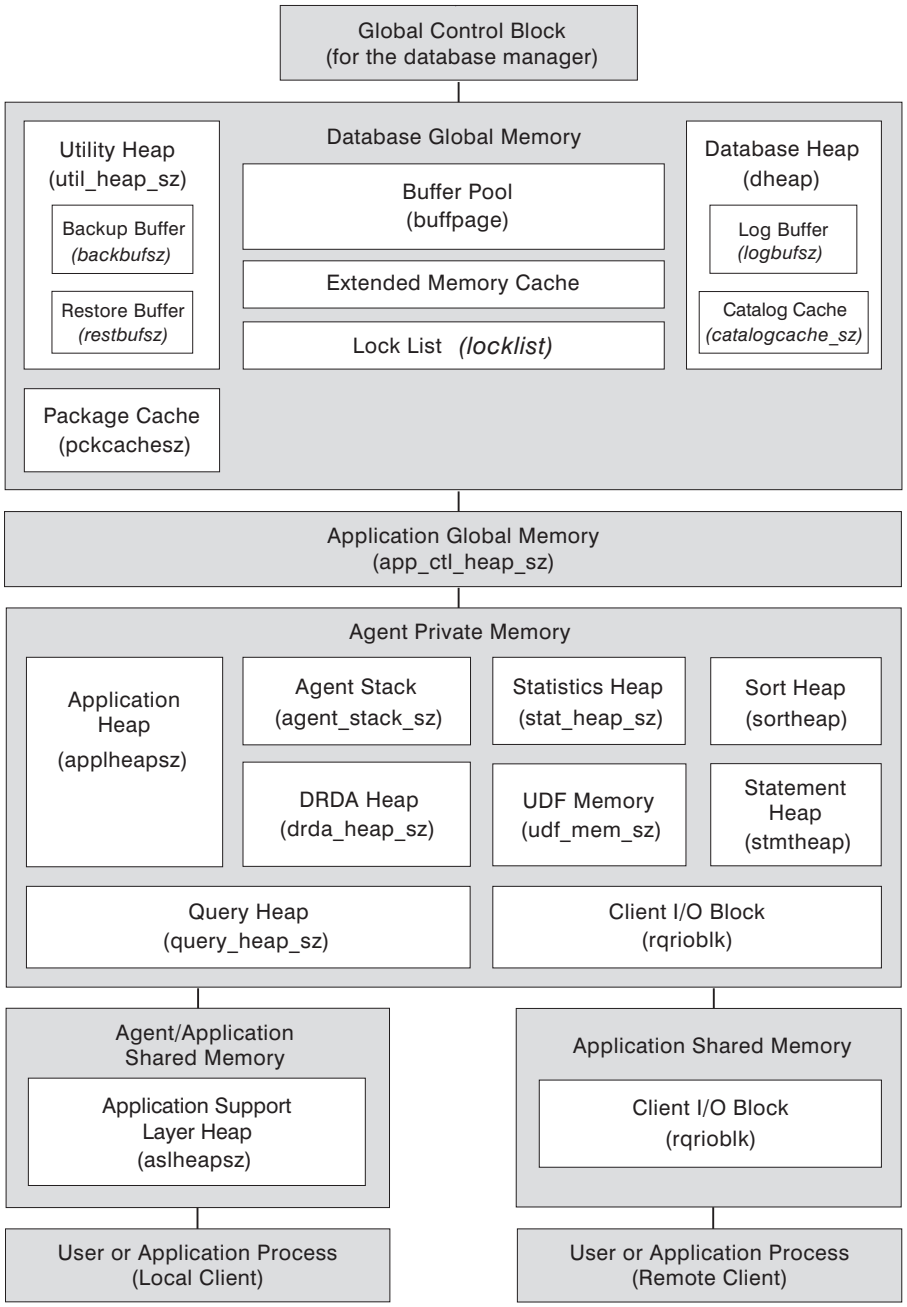


Figure 50. How memory is used by the database manager.

- The maximum size of memory segments is determined by the values of the following parameters:
  - "Buffer Pool Size (buffpage)" on page 470 (if a buffer pool size is -1), or the explicit sizes that were specified when the buffer pools were created or altered
  - "Maximum Storage for Lock List (locklist)" on page 477
  - "Database Heap (dbheap)" on page 472
  - "Utility Heap Size (util_heap_sz)" on page 475
  - "Extended Storage Memory Segment Size (estore_seg_sz)" on page 507
  - "Number of Extended Storage Memory Segments (num_estore_segs)" on page 507.

Agent Private Memory

- The number of memory segments is limited by the lower of:
  - The total of *maxappls* for all active databases (see "Maximum Number of Active Applications (maxappls)" on page 508)
  - The value of *maxagents* (see "Maximum Number of Agents (maxagents)" on page 514), and (for MPP/SMP systems) *max_coordagents* (see "Maximum Number of Coordinating Agents (max_coordagents)" on page 515).
- The maximum size of memory segments is determined by the values of the following parameters:
  - "Application Heap Size (applheapsz)" on page 484
  - "Sort Heap Size (sortheap)" on page 482
  - "Statement Heap Size (stmtheap)" on page 484
  - "Statistics Heap Size (stat_heap_sz)" on page 485
  - "Query Heap Size (query_heap_sz)" on page 485
  - "Client I/O Block Size (rqrioblk)" on page 493
  - "DRDA Heap Size (drda_heap_sz)" on page 486
  - "UDF Shared Memory Set Size (udf_mem_sz)" on page 487
  - "Agent Stack Size (agent_stack_sz)" on page 488.

Application Shared Memory

- The total number of agent/application shared memory segments (for local clients) is limited by the lower of:
  - The total of *maxappls* for all active databases (see "Maximum Number of Active Applications (maxappls)" on page 508)
  - The value of *maxagents* (see "Maximum Number of Agents (maxagents)" on page 514), or (for MPP/SMP systems) *max_coordagents* (see "Maximum Number of Coordinating Agents (max_coordagents)" on page 515).
- For MPP/SMP systems, space is also required for the application control heap, which is shared between the agents that are working on behalf of the same application at one database partition. The heap is allocated when the first agent to receive a request from the application requests a connection. The agent can be either a coordinating agent or a subagent (see "Database Agents" on page 417). The application

control heap size is determined by the database configuration parameter *app_ctl_heap_sz*.

Setting Parameters That Affect Memory Usage

Parameters that allocate memory should *never* be set at their highest values, even on systems with the maximum amount of memory installed, unless such a value has been carefully justified. Many of the parameters can allow the database manager to very easily and quickly take up all of the available memory on a machine. In addition, the management of a large amount of memory can take significant additional work on the part of the database manager and thus incur even more overhead.

Some UNIX-based operating systems allocate swap space when a process allocates memory and not when it is paged out to swap space. In these cases, you should ensure the total shared memory size is backed with the equivalent amount of paging space.

For most of the configuration parameters, memory is only committed as it is required. These parameters reflect the maximum size of a particular memory heap. The notable exceptions to this rule are the following parameters for which memory is fully committed based on the parameter value:

- "Buffer Pool Size (buffpage)" on page 470 (if a buffer pool size is -1), or the explicit sizes that were specified when the buffer pools were created or altered.
- "Maximum Storage for Lock List (locklist)" on page 477
- "Application Support Layer Heap Size (aslheapsz)" on page 492

The appropriate values for these types of parameters can best be determined by benchmarking, where typical and worst-case SQL statements are run against the server and the values of the parameters are modified until the point of diminishing return for performance is found. If performance versus parameter values were graphed, the point where the curve begins to plateau or decline would indicate the point at which additional allocation provides no additional value to the application and is therefore simply wasting memory. (See Chapter 18, "Benchmark Testing" on page 447.)

The upper limits of memory allocation for several parameters may be beyond the memory capabilities of existing hardware and operating systems. These limits were chosen to allow for future growth.

For valid parameter ranges, see the parameter descriptions in Chapter 19, "Configuring DB2" on page 459.

FCM Requirements

Start with default values when configuring the following Fast Communications Manager (FCM) configuration parameters:

- "Number of FCM Buffers (fcm_num_buffers)" on page 557
- "Number of FCM Request Blocks (fcm_num_rqb)" on page 558
- "Number of FCM Connection Entries (fcm_num_connect)" on page 558
- "Number of FCM Message Anchors (fcm_num_anchors)" on page 556

To tune these parameters, use the database system monitor to monitor the low water mark for the free buffers, free message anchors, free connection entries, and the free request blocks. If the low water mark is less than 10 percent of the number of the corresponding free data item, increase the value of the corresponding parameter. For information on the database system monitor, see “Using the Database System Monitor” on page 420.

For information on FCM, see “Enable FCM Communications” on page 70.

Managing the Database Buffer Pool

A buffer pool is an area of storage into which database pages (containing table rows or index entries) are temporarily read and changed. The purpose of the buffer pool is to improve database system performance. Data can be accessed much faster from memory than from a disk. Therefore, the fewer times the database manager needs to read from or write to a disk, the better the performance.

The configuration of one or more buffer pools is the single most important tuning area, since it is here that most of the data manipulation takes place for applications connected to the database (excluding large objects and long field data).

When an application accesses a row of a table for the first time, the database manager places the page containing that row in the buffer pool. The next time any application requests data, the buffer pool is checked first. If the requested data is found on pages kept in the buffer pool, the database manager does not need to go out to disk storage to retrieve the requested data. Avoiding the need to retrieve data from disk storage results in faster performance.

Pages stay in the buffer pool until the database is shut down, or until the space occupied by a page is required for another page. The space chosen in the buffer pool to bring in another page is selected using criteria such as the following:

- The last reference to a page
- The likelihood of the page being referenced again by the last agent that looked at the page
- The type of page
- Whether or not a page was changed in memory but not written out to disk. (Changed pages are always written to disk before being overwritten.)

Note: After changed pages are written out to disk, they are not removed from the buffer pool unless the space they occupy is needed for other pages. Until they are overwritten, they can be accessed again if their data is needed.

Pages in the buffer pool can have different attributes:

- In-use pages are currently being read or updated. They can be read, but not updated, by other agents.
- “Dirty” pages are pages where data has been changed but has not yet been written to disk. After a page is written to disk, it is considered “clean,” and remains in the

buffer pool. The space occupied by clean pages can be used for new pages, and is available for migration to an associated extended storage cache (if defined).

Pages can be written from the buffer pool to disk when the percentage of space occupied by changed pages in the buffer pool has exceeded the value specified by the *chngpgs_thresh* configuration parameter. You also may need to configure the database to include more than one page-cleaner agent. These agents write out changed pages to disk so that the database agents can find usable space in the buffer pool.

Page cleaner agents perform I/O that would otherwise have to be performed by the database agents. As a result, your applications can run faster, because transactions are not forced to wait while their database agents write pages to disk. (Page-cleaner agents are sometimes referred to as *asynchronous page cleaners* or *asynchronous buffer writers* because they can run in parallel with the database agents.)

To change the number of page-cleaner agents, use the *num_iocleaners* configuration parameter (the default is to create one page-cleaner agent). For information, see "Number of Asynchronous Page Cleaners (num_iocleaners)" on page 502.

Writing pages to disk also allows for faster recovery of the database should a system crash occur, because the database manager is able to rebuild more of the buffer pool from disk rather than having to use the database log files. As a result, page cleaning is requested if the size of the log that would need to be read during recovery exceeds the following maximum:

```
logfilsiz * softmax
```

where:

- *logfilsiz* represents the size of the log files (see "Size of Log Files (logfilsiz)" on page 519)
- *softmax* represents the percentage of log files to be recovered following a database crash (see "Recovery Range and Soft Checkpoint Interval (softmax)" on page 526).

  For example, if the value of softmax is 250, then 2.5 log files will contain the changes that need to be recovered if a crash occurs.

You may use the database system monitor to help you track the number of times that page cleaning is requested to minimize log read time during recovery. For more information see the *pool_lsn_gap_clns (buffer pool log space cleaners triggered)* monitor element description in the *System Monitor Guide and Reference* manual.

The size of the log that would need to be read during recovery is the difference between the location of the following in the log:

- The most recently written log record
- The log record that describes the oldest change to data in the buffer pool.

The following figure illustrates how the work of managing the buffer pool can be shared between page-cleaner agents and database agents, compared to the database agents performing all of the I/O.

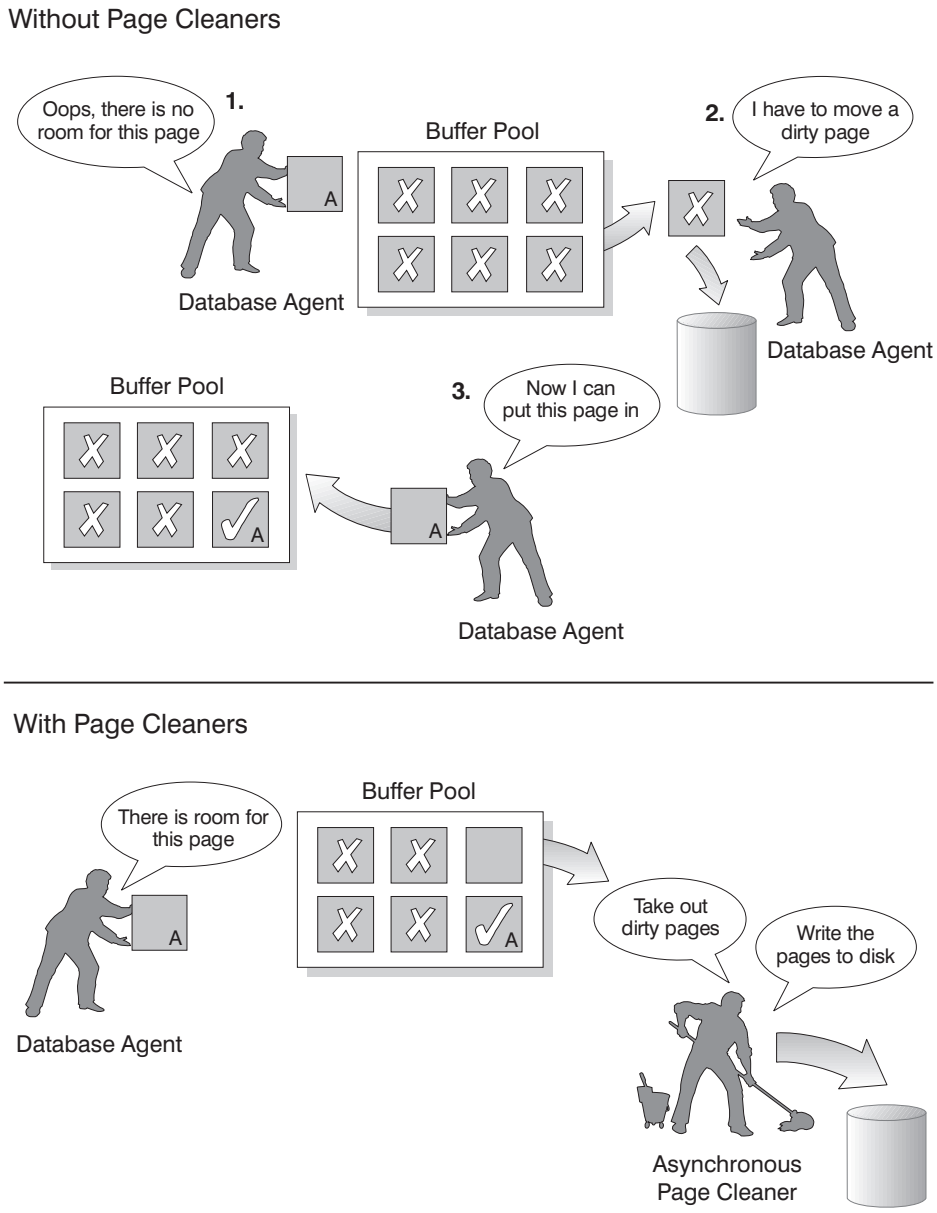


Figure 51. Asynchronous Page Cleaner. "Dirty" pages are written out to disk.

Managing Multiple Database Buffer Pools

Each database requires at least one buffer pool. However, depending on your needs you may choose to create several buffer pools, each of a different size, for a single database. The CREATE, ALTER, and DROP BUFFERPOOL statements allow you to create, change, or remove a buffer pool. You can specify which data is cached in a buffer pool with the CREATE TABLESPACE and ALTER TABLESPACE statements.

The *buffpage* configuration parameter specifies the size of any buffer pool, if the buffer pool's size is specified as -1 in the SYSIBM.SYSBUFFERPOOLS catalog table. (Otherwise this parameter is ignored.) A buffer pool's size can be set with the DDL statements ALTER BUFFERPOOL or CREATE BUFFERPOOL.

A new or migrated database has a default buffer pool called IBMDEFAULTBP. Migrated databases have a default buffer pool with a size determined by the *buffpage* configuration parameter (because the SIZE of the buffer pool in the SYSIBM.SYSBUFFERPOOLS catalog table is set to -1 for migration). New databases have a default buffer pool with a size determined by the platform. Once a database is created or migrated, then other buffer pools can be created for it.

In a partitioned database environment, each buffer pool for a database has the same default definition on all database partitions (unless it was otherwise specified in the CREATE BUFFERPOOL statement, or the buffer pool's size was changed for a particular database partition with the ALTER BUFFERPOOL statement).

Table spaces can be assigned to specific buffer pools with the BUFFERPOOL parameter for either the CREATE or ALTER TABLESPACE statements. When table spaces are created, if they are not specifically assigned to a buffer pool they are assigned to the default buffer pool.

When creating or altering buffer pools, the total memory that is required by all buffer pools must be available to the database manager so that all of the buffer pools can be allocated when the database is started. Should this memory not be available when a database is started, the database manager attempts to start the default buffer pool (IBMDEFAULTBP), but only with a minimal size. A warning message is returned with each failed attempt to start a buffer pool; the database continues in this operational state until its configuration is changed and the database can be fully restarted.

Note: Although the size and attributes associated with the default buffer pool can be changed, it cannot be dropped. Also, there is a minimum size for each buffer pool that is based on the platform being used.

There are advantages to having a large amount of memory allocated to buffer pools. For example, larger buffer pool sizes:

- Enable often-requested data pages to be kept in the buffer pool, allowing for quicker access. Fewer I/O operations can reduce I/O contention, thereby providing better response time and reducing the processor resource needed for I/O operations.

- Provide the opportunity to achieve higher transaction rates with the same response time.
- Prevent I/O contention for frequently used disk storage devices such as catalog tables and frequently referenced user tables and indexes. Sorts required by queries also benefit from reduced I/O contention on the disk storage devices containing the temporary table spaces.

Choosing One or Many Buffer Pools

If any of the following conditions apply to your system, you should use only a single buffer pool:

- The total buffer space is less than 10 000 4-KB pages.
- People with the application knowledge to do specialized tuning are not available.
- You are working on a test system.

If your system is not constrained by these conditions, then consider using more than one buffer pool for the following potential performance improvements:

- You can put temporary table spaces into a separate buffer pool to provide better performance for queries that require temporary storage, especially sort-intensive queries.
- If you have data that must be accessed repeatedly and quickly by many short update transaction applications, then you should consider moving the table space containing the data into a separate buffer pool. If this buffer pool is sized appropriately, its pages have a better chance of being found, contributing to a lower response time and a lower transaction cost.
- You can isolate data into separate buffer pools to favor certain applications, data, and indexes. For example, you might want to put tables and indexes that are updated frequently into a buffer pool that is separate from those tables and indexes that are frequently queried but infrequently updated. This change will reduce the impact of the frequent updates (on the first set of tables) on the frequent queries (on the second set of tables).
- You can use smaller buffer pools for the data accessed by applications that are seldomly used, especially in the case where an application requires very random access into a very large table. In such a case, there is no need to keep the data in buffer pool memory for longer than a single query. It is better to keep a small buffer pool for this data, and free up the extra memory for other uses (for example, for other buffer pools).
- After separating different activities and data into separate buffer pools, good and relatively inexpensive performance diagnosis data can be produced from statistics and accounting traces.

Prefetching Data into the Buffer Pool

Prefetching index and data pages into the buffer pool can help improve performance by reducing the time spent waiting for I/O to complete. To *prefetch* pages means that one

or more pages are retrieved from disk in anticipation of their use. There are two categories of prefetch:

- *Sequential prefetch* is a mechanism that reads consecutive pages into the buffer pool before the pages are required by the application. (See “Understanding Sequential Prefetching.”)
- *List prefetch*, or list sequential prefetch, is a way to access data pages efficiently, even when the data pages needed are not consecutive. (See “Understanding List Prefetching” on page 407.)

These two methods of reading data pages are in addition to a normal read. A normal read is used when just one or a few consecutive pages are retrieved. During a normal read, one page of data is transferred.

For further information on enabling prefetching, see also “Configuring I/O Servers for Prefetching and Parallel I/O” on page 408.

Understanding Sequential Prefetching

Reading several consecutive pages into the buffer pool using a single I/O operation can greatly reduce the overhead associated with running your application. In addition, performing multiple I/O operations in parallel to read in several ranges of pages at the same time can help reduce the time your application needs to wait for I/O operations to complete.

Prefetching is started when the database manager determines that sequential I/O is appropriate and that prefetching may help to improve performance. In cases such as table scans and table sorts, the database manager can easily determine that sequential prefetch will improve I/O performance. In these cases, the database manager automatically starts sequential prefetch. The following example could require a table scan and would be a good candidate for sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
```

The number of pages that the database manager will prefetch can be defined for each table space using the PREFETCHSIZE clause with either the CREATE TABLESPACE or ALTER TABLESPACE statements. The value specified is maintained in the PREFETCHSIZE column of the SYSCAT.TABLESPACES system catalog table.

It is a good practice to explicitly set the PREFETCHSIZE value as a multiple of the EXTENTSIZE value for your table space and the number of table space containers. (The extent size is the number of pages that the database manager writes to a container before using a different container; see “Designing and Choosing Table Spaces” on page 38.) For example, if the extent size is 16 pages and the table space has two containers, you could choose to set the prefetch quantity to 32 pages.

The database manager monitors buffer pool usage to ensure that prefetching of data does not remove pages from the buffer pool if those pages are needed by another unit of work. To avoid problems, the database manager may choose to limit the number of pages being prefetched to a quantity less than you specified for the table space.

The setting of the prefetch size can have significant performance implications, particularly for large table scans. You can use the database system monitor and other system monitor tools to help you tune PREFETCHSIZE for your table spaces. For example, you can gather information about whether:

- There are I/O waits for your query, using monitoring tools available for your operating system.
- Prefetch is occurring, by looking at the *pool_async_data_reads (buffer pool asynchronous data reads)* data element provided by the database system monitor. See the *System Monitor Guide and Reference* for more information.

If there are I/O waits and the query is prefetching data, you can try increasing the value of PREFETCHSIZE. It is possible that the prefetcher is not the cause of the I/O wait, in which case increasing the PREFETCHSIZE value will not improve the performance of your query.

In all types of prefetch, multiple I/O operations may be performed in parallel when the prefetch size is a multiple of the extent size for the table space and the extents of the table space are in separate containers. For better performance the containers should be configured to use separate physical devices. For more information on parallel prefetching, see “Configuring I/O Servers for Prefetching and Parallel I/O” on page 408.

Understanding Sequential Detection

There are cases for which it is not immediately obvious whether sequential prefetch will improve performance. In these cases, the database manager can monitor I/O and if sequential page reading is occurring the database manager can activate prefetching. Prefetching in this case can be activated and deactivated by the database manager when it deems it appropriate. This type of sequential prefetch is known as *sequential detection* and applies to both index and data pages. You may use the *seqdetect* configuration parameter (see “Sequential Detection Flag (seqdetect)” on page 505) to control whether the database manager should perform sequential detection. If sequential detection is turned on, it could determine that the following SQL statement would benefit from sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
WHERE EMPNO BETWEEN 100 AND 3000
```

In this example, the optimizer may have chosen to scan the table using an index on the EMPNO column. If the table is highly clustered with respect to this index, then the data page reads will be almost sequential and prefetching may improve performance. In this case, data page prefetch will occur.

Index page prefetch may also occur in this example. If a large number of index pages have to be examined and the database manager detects that sequential page reading of the index pages is occurring, then index page prefetching will occur.

Understanding List Prefetching

List prefetch, or list sequential prefetch, is a way to access data pages efficiently, even when the data pages needed are not contiguous. List prefetch can be used in conjunction with either single or multiple index access.

Prefetching and Intra-Partition Parallelism

Prefetching is very important to the performance of intra-partition parallelism, which uses multiple subagents when scanning an index or a table. These parallel scans introduce larger data consumption rates, which require higher prefetch rates.

The cost of inadequate prefetching is higher for parallel scans than serial scans. If prefetching does not occur when executing a serial scan, the query runs more slowly because the agent always needs to wait for I/O. If prefetching does not occur when executing a parallel scan, all subagents may need to wait for one subagent that is waiting for I/O.

Because of its importance, prefetching is performed more aggressively with intra-partition parallelism. The sequential detection mechanism tolerates larger gaps between adjacent pages so that the pages can be considered sequential. The width of these gaps increases with the number of subagents involved in the scan.

Configuring I/O Servers for Prefetching and Parallel I/O

To enable prefetching, the database manager starts separate threads of control, known as *I/O servers*, to perform page reading. As a result, the query processing is divided into two parallel activities: data processing (CPU) and data page I/O. The I/O servers wait for prefetch requests from the CPU processing activity. These prefetch requests contain a description of the I/O needed to satisfy the anticipated data needs. The reason for prefetching determines when and how the database manager generates the prefetch requests. (See “Understanding Sequential Prefetching” on page 406 and “Understanding List Prefetching” on page 407 for more information.)

The following figure illustrates how I/O servers are used to prefetch data into a buffer pool.

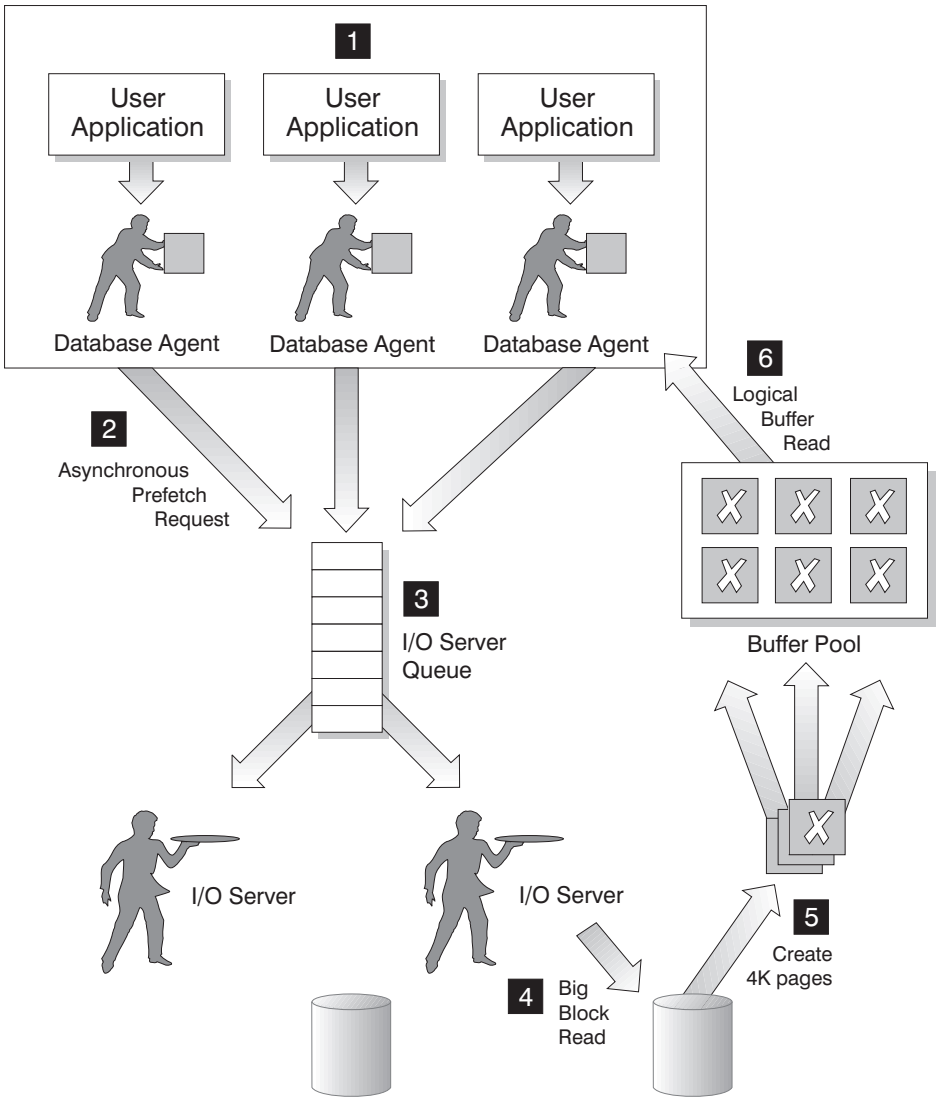


Figure 52. Prefetching Data using I/O Servers

The following steps are illustrated in Figure 52:

1. The user application passes the SQL request to the database agent.
2. The database agent determines that prefetching should be used to obtain the data required to satisfy the SQL request and writes a prefetch request to the I/O server queue.

3, **4** The first available I/O server will read the prefetch request from the queue and read the data from the table space into the buffer pool. Depending on the number of prefetch requests in the queue and the number of I/O servers configured by the *num_ioservers* configuration parameter, multiple I/O servers can be fetching data from the table space at the same time.

5 The database agent performs the necessary actions against the data pages in the buffer pool in order to return the result of the SQL request back to the user application.

Configuring enough I/O servers with the *num_ioservers* configuration parameter can greatly enhance the performance of queries for which prefetching of data can be used. Having some extra I/O servers configured will not hurt performance because extra I/O servers are not used and their memory pages will get paged out. Each I/O server process is numbered and the database manager will always use the lowest numbered process that is available and, as a result, some of the upper numbered processes may never be used.

To determine the number of I/O servers that you should configure, consider the following:

- The amount of concurrent activity against the database. That is, the number of database agents that could be writing prefetch requests to the I/O server queue at any given time.
- The highest degree to which the I/O servers can work in parallel. For more information, see "Enabling Parallel I/O."

Enabling Parallel I/O

For situations in which multiple containers exist for a table space, the database manager can initiate *parallel I/O*. Parallel I/O refers to the ability of the database manager to use multiple I/O servers to process the I/O requirements of a single query. Each I/O server is assigned the I/O workload for a separate container, allowing several containers to be read in parallel. Performing I/O in parallel can result in significant improvements to I/O throughput.

While a separate I/O server will handle the workload for each container, the actual number of I/O servers that can perform I/O in parallel will be limited to the number of physical devices over which the requested data is spread. This also means you need as many I/O servers as the number of physical devices.

How parallel I/O is initiated and used is dependent on the reason for performing the I/O:

- **Sequential prefetch**

  For sequential prefetch, parallel I/O is initiated when the prefetch size is a multiple of the extent size for a table space. Each prefetch request is then broken into multiple, smaller, requests along the extent boundaries. These smaller requests are then assigned to different I/O servers.

- **List prefetch**

For list prefetch, each list of pages is divided into smaller lists according to the container in which the data pages are stored. These smaller lists are then assigned to different I/O servers.

- **Database or table space backup and restore**

  For backing up or restoring data, the number of parallel I/O requests are equal to the backup buffer size divided by the extent size up to a maximum value equal to the number of containers.

- **Database or table space restore**

  For restoring data, the parallel I/O requests are initiated and split in a manner that is the same as that used for sequential prefetch. Instead of restoring the data into the buffer pool, the data is moved directly from the restore buffer to disk.

- **Load**

  When loading data you can specify the level of I/O parallelism with the LOAD command's DISK_PARALLELISM option. (If it is not specified, a default is used based on the cumulative number of table space containers for all table spaces associated with the table.)

For optimal performance of parallel I/O, ensure that:

- There are enough I/O servers. You should configure the number of I/O servers to be slightly higher than the number of containers used for all table spaces within the database.
- The extent size and prefetch size are sensible for the table space. Prefetch size should not be too large, to prevent over-use of the buffer pool. (An ideal size is a multiple of the extent size and the number of table space containers.) The extent size should be fairly small, with a good value being in the range of 8 to 32 pages.
- The containers are configured to reside on separate physical drives.
- All containers are the same size to ensure a consistent degree of parallelism.

  If one or more containers are smaller than the others, they will reduce the potential for optimized parallel prefetch. For example:

  – After a smaller container is filled up, additional data is stored in the remaining containers, causing the containers to become unbalanced. Unbalanced containers reduce the performance of parallel prefetching, because the number of containers from which data can be prefetched may be less than the total number of containers.

  – If a smaller container is added at a later date and the data is rebalanced, the smaller container will contain less data than the other containers. Its small amount of data relative to the other containers will not optimize parallel prefetching.

  – If one container is larger and all of the other containers fill up, it will be the only container to store additional data. The database manager will not be able to use parallel prefetch to access this additional data.

- There is adequate I/O capacity when using intra-partition parallelism. Intra-partition parallelism can be used on SMP machines to reduce a query's elapsed time by running the query on multiple processors. Sufficient I/O capacity is required to keep each processor busy, usually requiring additional physical drives to provide the I/O capacity.

  Prefetching must occur at higher rates to use I/O capacity effectively. The prefetch size should be higher for prefetching to occur at higher rates. The prefetch size should be a multiple of the extent size and the number of table space containers. Ideally, containers should be configured to reside on separate physical drives.

  The number of physical drives required could depend on the speed and capacity of the drives and the I/O bus, and on the speed of the processors.

Allocating Multiple Pages at a Time

The database manager expands the database as needed when the *multipage_alloc* database configuration parameter is set to YES. (See "MultiPage File Allocation Enabled (multipage_alloc)" on page 540.) The default value for this parameter is NO, causing the database manager to expand the database file one page at a time.

To set *multipage_alloc* to YES (and gain the associated performance improvement), use the **db2empfa** tool in the sqllib/bin directory. (For more information on this tool, see the *Command Reference*.)

Sorting

Sorting is often required for a query, and the proper configuration of the sort heap areas can be crucial to the query's performance. Sorting is required when:

- No index exists to satisfy a requested ordering (for example a SELECT statement that uses the ORDER BY clause)
- An index exists but sorting would be more efficient than using the index
- Creating an index (if the *indexsort* configuration parameter is set to yes).

Different Types of Sorting

Sorting involves two steps:

1. A sort phase
2. Return of the results of the sort phase.

How the sort is handled within these two steps results in different categories or types by which we can describe the sort. When considering the sort phase, the sort can be categorized as "overflowed" or "non-overflowed." When considering the return of the results of the sort phase, the sort can be categorized as "piped" or "non-piped."

Overflowed and Non-Overflowed

If the information being sorted cannot fit entirely into the sort heap (a block of memory that is allocated each time a sort is performed) it overflows into temporary database tables. Sorts that do not overflow always perform better than those that do.

Piped and Non-Piped

If sorted information can return directly without requiring a temporary table to store a final, sorted list of data, it is referred to as a "piped sort." If the sorted information requires a temporary table to be returned, it is referred to as a "non-piped sort." A piped sort always performs better than a non-piped sort.

The following combinations of the two sort categories are possible:

- A non-overflowed, piped sort (the best performing)
- A non-overflowed, non-piped sort
- An overflowed, piped sort
- An overflowed, non-piped sort (the worst performing)

Tuning the Parameters that Affect Sorting

The following situations affect the performance of sorting:

- The settings for the following configuration parameters:

  "Sort Heap Size (sortheap)" on page 482

  Specifies the amount of memory to be used for each sort

  "Sort Heap Threshold (sheapthres)" on page 482

  Controls the total amount of memory for sorting available across the entire instance for all sorts.

- Statements that involve a large amount of sorting
- Missing indexes that could help avoid unnecessary sorting
- Application logic that does not minimize sorting
- Parallel sorting, which improves the performance of sorts but can only occur if the statement uses intra-partition parallelism (see "Enabling Parallel I/O" on page 410).

Looking for Indicators of Sorting Performance Problems

To tell if you have an overall problem with sorting, look at the total CPU time spent sorting compared to the time spent on the whole application. The database system monitor can help (see "Using the Database System Monitor" on page 420). In particular, the Performance Monitor (know by "Snapshot Monitor" and "Event Monitor"), available from the Control Center on the Windows 95, Windows NT, and OS/2 operating systems, shows *total sort time* by default, along with other times such as *I/O* and *lock wait*.

If total sort time is a large proportion of the other times then look at the following values, which are also shown by default:

Percentage of overflowed sorts

This variable (on the performance details view of the Snapshot Monitor) shows the percentage of sorts that overflowed. If the percentage of overflowed sorts is high, increase the *sortheap* and/or *sheapthres* configuration parameters if there were any post-threshold sorts. (To determine if there were any post threshold sorts, use the Snapshot Monitor.)

Post threshold sorts

If post threshold sorts are high, increase *sheapthres* and/or decrease *sortheap*.

In general, make the overall sort memory available across the instance (*sheapthres*) as large as possible without causing excessive paging. It is possible for a sort to be done entirely in sort memory. However, if this causes the operating system to perform excessive page swapping to accommodate that sort memory you can lose the advantage of a large sort heap. So, whenever you adjust the sorting configuration parameters, use an operating system monitor to track any changes in system paging.

Also note that in a piped sort, the sort heap does not get freed until the application closes the cursor associated with that sort. So a piped sort can use up memory until the cursor is closed.

Techniques for Managing Sorting Performance

You can use the database system monitor and benchmarking techniques to help set the *sortheap* and *sheapthres* configuration parameters. Do the following for each database manager and its databases:

- Set up and run a representative workload.
- For each applicable database, collect average values for the following performance variables over the benchmark workload period:
  - Total sort heap in use
  - Active sorts

  These performance variables are shown on the performance details view of the Snapshot Monitor.
- Set *sortheap* to the average *total sort heap in use* for each database.
- Take the largest *sortheap* value for all databases in a database manager, and set *sheapthres* to the average number of active sorts multiplied by the average size of the sort heap. This is a recommended initial setting. You can then use benchmark techniques to refine this value.

You can also identify particular applications and statements where sorting is a significant performance problem:

- Set up event monitors at the application and statement level to help you identify applications with the longest total sort time.
- Within each of these applications, find the statements with the longest *total sort time*.
- Tune these statements using a tool such as Visual Explain.
- Ensure that appropriate indexes exist. You can use Visual Explain to identify all the sort operations for a given statement. Then investigate whether or not an appropriate index exists for each table accessed by the statement.

Note: You can search through the explain tables to identify which queries have sort operations. (See Appendix L, “SQL Explain Tools (db2expln and dynexpln)” on page 783.)

Reorganizing Table Data

The performance of SQL statements that use indexes can be impaired after many updates, deletes, or inserts have been made. Generally, newly inserted rows cannot be placed in a physical sequence that is the same as the logical sequence defined by the index. This means that the database manager must perform additional read operations to access the data, because logically sequential data may be on different physical data pages that are not sequential.

In general, reorganizing a table takes more time than running statistics. Performance may be improved sufficiently by obtaining the current statistics for your data and rebinding your applications, so try this first. If this does not improve performance, the data in the tables and indexes may not be arranged efficiently, so reorganization may help. The information in this section applies not only to reorganizing your own tables, but also to the system catalog tables which may also require reorganization.

The REORGCHK command returns information about the physical characteristics of a table, and whether or not it would be beneficial to reorganize that table. This command can be used through the command line processor. See the *Command Reference* for more information, including how to interpret the command output.

The REORG utility optionally rearranges data into a physical sequence according to a specified index. REORG has an option to specify the order of rows in a table with an index, thereby clustering the table data according to the index and improving the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility. As a result, SQL statements requiring rows in the indexed order can be processed more efficiently. REORG also stores the tables more compactly by removing unused, empty space.

You may wish to consider the following factors to determine when to reorganize your table data:

- The volume of insert, update, and delete activity
- Any significant change to the performance of queries which use an index with a high cluster ratio
- Running statistics (RUNSTATS) does not improve the performance of queries
- The REORGCHK command indicates a need to reorganize your table
- The cost of reorganizing your table, including the CPU time, the elapsed time, and the reduced concurrency resulting from the REORG utility locking the table until the reorganization is complete.

To execute the REORG utility, you must have SYSADM, SYSMAINT, SYSCTRL or DBADM authority, or CONTROL privilege on the table.

The REORG utility uses temporary tables that can be significantly larger than the original table, if columns were added to a table, or a table has LOB columns. If these temporary tables are larger, the resulting permanent table, created by the REORG utility, will also be larger.

The REORG utility allows you to specify a temporary table space, which is used to create the temporary REORG table. If a temporary table space is not specified, the REORG utility will create the temporary REORG tables in the table space that contains the table being reorganized. The following guidelines can assist you in determining whether to use a temporary table space:

- It is generally recommended that you specify a temporary SMS table space.
- Do not specify a temporary table space if you think that the REORG table will fit in the same DMS table space as the base table. In this case, the REORG utility will operate much faster than if a temporary table space was specified, but the table space will need enough available free space for a second copy of the table. This second copy could be smaller or larger than the original table, depending on how much unused space exists in the original table and on whether the reorganization will expand the size of LOBs. Using the same DMS table space also increases the amount of space required for logging, because a log record is written for each extent consumed by the reorganized table.
- Using a temporary DMS table space is generally not recommended since you can only have one REORG in progress using this type of table space.

If the REORG utility does not complete successfully, do **not** delete any temporary files, tables or table spaces. These files and tables are used by the database manager to roll back the changes made by the REORG utility, or to complete the reorganization, depending on how far the reorganization had progressed before the failure.

In a partitioned database, the REORG utility reorganizes data on each partition. If the utility fails on any partition, only the failing partition is rolled back. If you specify a directory path to store temporary tables, this path is extended by the database manager at each database partition. Therefore, if you specify a path that is shared by other database partitions, the temporary files are stored in different subdirectories (identified by node name) under this path.

Performance Considerations for DMS Devices

If you are using Database Managed Storage (DMS) device containers for your table spaces, you need to understand the following so you can effectively administer your environment:

- **Buffering of data**

  Table data read from disk is normally available in the database's buffer pool (see “Managing the Database Buffer Pool” on page 401). In some cases, a data page can be freed from the buffer pool before the application has actually used that page. (This can happen if the buffer pool space is required for other data pages.) For table spaces using system managed storage (SMS), when the database manager subsequently requests the page from the file system, the file system may still have that page in its own cache. This can eliminate I/O that would otherwise have been required.

  Table spaces using database managed storage (DMS) device containers do **not** use the file system or its cache. As a result, you may wish to increase the size of

the database buffer pool and reduce the size of the file system cache to offset the fact that double buffering is not being done with DMS table spaces that use device containers.

If you notice, through the use of system-level monitoring tools, that I/O is higher for a DMS table space using device containers compared to the equivalent SMS table space, this difference could be due to the double buffering discussed above.

- **Using LOB or LONG data**

  When an application retrieves either LOB or LONG data, the database manager does not use its buffers to cache the data. Every time an application needs one of these pages, the database manager must retrieve it from disk.

  If LOB or LONG data is stored in SMS or DMS-file table spaces, then there is a chance that the operating system may cache the pages in main memory and thus it will be accessed more quickly (because it is faster to retrieve pages from main memory than from disk).

  Because system catalogs contain some LOB columns, it is recommended that you keep them in SMS (or alternatively in DMS-file) table spaces.

Managing Initialization Overhead

The ACTIVATE DATABASE command starts up selected databases. Using this command in a partitioned database results in an attempt to activate the selected partitioned database on all database partitions. By using this command, no application time is spent on database initialization or startup.

Databases that you have initialized using the ACTIVATE DATABASE command must be shut down with the DEACTIVATE DATABASE command or with the *db2stop* command; the last application disconnecting from the database will not shut it down. For more information on the ACTIVATE and DEACTIVATE commands, refer to the *Command Reference* manual.

If a database has not been started, and a CONNECT TO (or an implicit connect) is encountered in an application, then the application must wait while the database manager starts up the required database before it can do any work with that database. This is a startup cost that is borne by the first application to access a particular database. In a partitioned database, this startup cost is incurred on each database partition. Once the database is started, all other applications can connect to and use the database without a time cost associated with the database startup.

Database Agents

DB2 servers must facilitate communication between the database manager and client and local applications. UNIX-based environments use an architecture based on *processes*. For example, the DB2 communications listeners are created as processes. Intel operating systems such as OS/2 and Windows NT use an architecture based on *threads* to maximize performance. For example, the DB2 communications listeners are created as threads within the DB2 server's system controller process. For each

database being accessed, various processes/threads are started to deal with the various database tasks (for example, prefetching, communication, and logging).

One of the most crucial processes/threads are those of database agents, which facilitate the operations of applications with databases. Each process/thread of a client application has a single *coordinator agent* that operates on a database. Once the coordinator agent is created, it performs all database requests on behalf of its application, and communicates to other agents using inter-process communications (IPC) or remote communication protocols. Each agent operates with its own private memory and shares database manager and database global resources such as the buffer pool with other agents.

In symmetric multiprocessor (SMP) environments, partitioned database environments, and non-partitioned database environments in which the *intra_parallel* database manager configuration parameter is enabled, the coordinator agent distributes database requests to *subagents*, and these agents perform the requests for the application. Once the coordinator agent is created, it handles all database requests on behalf of its application by coordinating the subagents that perform requests on the database.

When a client disconnects from a database or detaches from an instance the coordinating agent will be:

- Freed and marked as idle, if the maximum number of pool agents has not been reached
- Terminated and its storage freed, if the maximum number of pool agents has been reached.

When idle, agents are not performing work on behalf of any applications, are waiting to be assigned, and reside in an *agent pool*. These agents are available for requests from coordinator agents operating on behalf of client programs, or for subagents operating on behalf of existing coordinator agents. The number of available agents is dependent on the database manager configuration parameters *maxagents* and *num_poolagents*.

If no idle agents exist when an agent is required, a new agent must be dynamically created. Creating a new agent involves a certain amount of overhead and as a result, improved CONNECT and ATTACH performance can be noticed if there is an idle agent that can be activated for a client.

When a subagent is working on behalf of an application, it is considered to be *associated* with that application. After completing the assigned work, it may be placed in the agent pool, but it remains associated with the original application. When the application requests additional work, the database manager first checks the idle pool for associated agents when finding an agent to work for the application.

For SMP/MPP systems, each partition (that is, each database server or node) has its own pool of agents from which subagents are drawn. Because of this pool, subagents do not have to be created and destroyed each time one is needed or is finished its work. The subagents can remain as associated agents in the pool and be used by the database manager for new requests from the application they are associated with.

The following database manager configuration parameters affect the number of database agents:

- "Maximum Number of Agents (maxagents)" on page 514. Once the number of agents reaches this value, all subsequent requests that require a new agent are denied until the number of agents falls below the value. This value applies to the total number of agents, whether coordinating agents or subagents, that are working on all applications.
- "Agent Pool Size (num_poolagents)" on page 515. The number of agents in the agent pool cannot exceed this value.
- "Initial Number of Agents in Pool (num_initagents)" on page 516. When the database manager is started, a pool of idle agents is created based on this value. This speeds up performance for initial queries.
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 515. For MPP/SMP systems, this value limits the number of coordinating agents.
- "Maximum Number of Concurrent Agents (maxcagents)" on page 513 . This value controls the number of *tokens* permitted by the database manager. For each database transaction (unit of work) that occurs when a client is connected to a database, a coordinating agent must obtain permission to process the transaction (known as a processing token) from the database manager. Only agents with a processing token are permitted by the database manager to execute a unit of work against a database. If a token is not available, the agent will wait until one is available, at which time the requested unit of work will be processed.

For MPP/SMP systems, the impact to performance and memory costs within the system is strongly related to how your agent pool is tuned:

- The database manager configuration parameter for agent pool size (*num_poolagents*) affects the total number of subagents that can be kept associated with applications on a partition (that is, node). If the pool size is too small (and the pool is full), a subagent will disassociate itself from the application it worked on and terminate. This situation leads to poor performance, because subagents must be constantly created and reassociated to applications.

  In addition, if the value of *num_poolagents* is too small, one application may fill the pool with associated subagents. Thus, when another application requires a new subagent and has no subagents in its associated agent pool, it will "steal" subagents from the agent pools of other applications. This situation is rather costly, and causes poor performance.
- The above situations must be weighed against the resource costs of allowing too many agents to be active at any given time.

  For example, if the value of *num_poolagents* is too large, associated subagents may sit unused in the pool for long periods of time. These subagents use database manager resources that will not be available for other tasks.

In addition to the database agents, there are other asynchronous activities performed by the database manager which run as their own process (or thread), including:

- Database I/O servers (or I/O prefetchers) (see “Prefetching Data into the Buffer Pool” on page 405)
- Database asynchronous page cleaners (see “Managing the Database Buffer Pool” on page 401)
- Database loggers
- Database deadlock detectors
- Event monitors
- Communication and IPC listeners
- Table space container rebalancers.

For more information on identifying the various DB2 processes, see the *Troubleshooting Guide*.

Using the Database System Monitor

The DB2 database manager maintains data about its operation, its performance, and the applications using it. This data is maintained as the database manager runs, and can provide important performance and troubleshooting information. For example, you can find out:

- The number of applications connected to a database, their status, and which SQL statements each application is executing, if any.
- Information that shows how well the database manager and database are configured, and helps you to tune them.
- When deadlocks occurred for a specified database, which applications were involved, and which locks were in contention.
- The list of locks held by an application or a database. If the application cannot proceed because it is waiting for a lock, there is additional information on the lock, including which application is holding it.

Because collecting some of this data introduces overhead on the operation of DB2, **monitor switches** are available to control which information is collected. To set monitor switches explicitly, use the UPDATE MONITOR SWITCHES command or the sqlmon() API. (You must have SYSADM, SYSCTRL, or SYSMAINT authority.)

There are two ways to access the data maintained by the database manager:

- **Taking a snapshot**

  Use the GET SNAPSHOT command from the command line; the Control Center on the OS/2, Windows 95, or Windows NT operating systems for a graphical interface; or write your own application, using the sqlmonss() API call.

  The Control Center, available from the DB2 folder or with the `db2cc` command, provides a performance monitor tool that samples monitor data at regular intervals by taking snapshots. This graphical interface provides either graphs or textual views of the snapshot data, in both detail and summary form. You can also define performance variables using data elements returned by the database monitor.

The Control Center's Snapshot Monitor tool also allows you to define exception conditions by specifying threshold values on performance variables. When a threshold value is reached, you can predefine any of the following actions to occur: notification through a window or audible alarm, logging of a record in a database, and/or execution of a script or program.

- **Using an event monitor**

  An event monitor captures system monitor information after particular events have occurred, such as the end of a transaction, the end of a statement, or the detection of a deadlock. This information can be written to files or to a named pipe.

  To use an event monitor:

  1. Create its definition with the Control Center or the SQL statement CREATE EVENT MONITOR. This statement stores the definition in database system catalogs.
  2. Activate the event monitor through the Control Center, or with the SQL statement:

     ```
     SET EVENT MONITOR evname STATE 1
     ```

     If writing to a named pipe, start the application reading from the named pipe before activating the event monitor. You can either write your own application to do this, or use **db2evmon**. Once the event monitor is active and starts writing events to the pipe, **db2evmon** will read them as they are being generated and write them to standard output.
  3. Read the trace. If using a file event monitor, you can view the binary trace that it creates in either of the following ways:
     - Use the **db2evmon** tool to format the trace to standard output.
     - Click on the **Event Analyzer** icon in the Control Center (on the Windows 95, Windows NT, or OS/2 systems) to use a graphical interface to view the trace, search for keywords, and filter out unwanted data.

For information on the system database monitor and the event monitor, see the *System Monitor Guide and Reference*. For a scenario of how to use them from the Control Center, see the *Administration Getting Started*.

Extending Memory

Your machine may have more real memory than the maximum amount of addressable memory (for example, addressable memory is usually between 2 GB and 4 GB on most platforms). You can configure any additional memory beyond addressable memory as an *extended storage cache*. Such an extended storage cache can be used by any of the defined buffer pools and should improve the performance of the database manager. The extended storage cache is defined in terms of memory segments.

DB2 makes use of addressable memory in your machine with buffer pools (see “Managing the Database Buffer Pool” on page 401). The extended storage cache is used by the buffer pools as a secondary level of caching (with the buffer pools

performing the first level of caching). Ideally buffer pools can hold the data that is most frequently accessed, while the extended storage cache can hold data that is accessed, but less frequently.

The following database configuration parameters influence the amount and the size of the memory available for extended storage:

- *num_estore_segs* defines the number of extended storage memory segments. The default for this configuration parameter is zero, which specifies that no extended storage cache exists. (See "Number of Extended Storage Memory Segments (num_estore_segs)" on page 507.)
- *estore_seg_sz* defines the size of each extended memory segment. This size is limited by the platform on which the extended storage cache is being used. (See "Extended Storage Memory Segment Size (estore_seg_sz)" on page 507.)

Because an extended storage cache is an extension to a buffer pool, it must always be associated with one or more specific buffer pools. Therefore, you must declare which buffer pools can take advantage of a cache once it is created. The CREATE and ALTER BUFFERPOOL statements have the attributes NOT EXTENDED STORAGE and EXTENDED STORAGE that control cache usage. By default neither IBMDEFAULTBP nor any newly created buffer pool will use extended storage.

The database manager cannot directly manipulate data that resides in the extended storage cache. However, it can transfer data from the extended storage cache to the buffer pool much faster than from disk storage.

When a row of data is needed from a page in an extended storage cache, the entire page is read into the corresponding buffer pool. If the row is changed while in the buffer pool, the page is **not** written back to the extended storage cache until it has been written to disk storage. The extended storage cache holds only pages that have been read into the buffer pool and have been discarded; they are kept in case they are needed again.

A buffer pool and its associated extended storage cache, if defined, are created when a database is activated or first connected to.

Chapter 15. Using the Governor

You use the governor to monitor and change the behavior of applications that run against a database.

The governor consists of two parts:

- A front-end utility
- A daemon

When you start the governor, you issue a start command from the governor front-end utility, which then starts the governor daemon. By default, a daemon is started on every partition in a partitioned database, but you can also use the front-end utility to start a single daemon at a specific partition to monitor the activity against the database partition found there. Or, a daemon can monitor the activity on a single-partition database. See “Starting and Stopping the Governor” for details.

Each governor daemon collects statistics about the applications running against a database. It then checks these statistics against the rules that you specified in a governor configuration file that applies to that specific database. (See “Creating the Governor Configuration File” on page 426 for details.) The governor then acts according to these rules. For example, a rule may indicate that an application is using too much resource. In this situation, the governor may change the application's priority or force it off the database, according to the instructions you specified in the governor configuration file.

If the action associated with a rule is to change the application's priority, the governor changes the priority of agents on the database partition on which the governor detected the resource violation. If the action associated with a rule is to force an application, the application is forced even if the governor that detected the resource violation is running on the application's coordinator node or in a partitioned database environment.

The governor also logs any actions that it takes. You can query the log files to review the actions that the governor has taken. For details, see “Governor Log Files” on page 432 and “Querying Governor Log Files” on page 433.

Starting and Stopping the Governor

You use the `db2gov` governor front-end utility to start or stop the governor (on either all database partitions or on a single database partition). You require SYSCTRL authority to use the utility.

The syntax for `db2gov` is as follows:

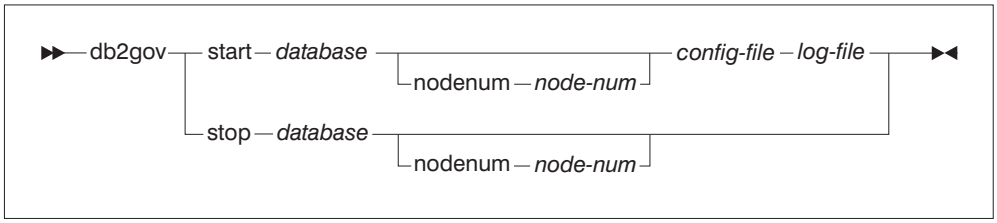


Figure 53. Syntax for db2gov

The parameters are as follows:

start *database*
: Starts the governor daemon to monitor the specified database. For *database*, you can specify either the database name or the database alias.

  The database name you specify must be the same name as that specified in the governor configuration file. The governor checks these two names to ensure that you are using the correct configuration file. If the front-end utility is started with one alias name and the governor configuration file contains a different alias, an error is reported because the governor cannot determine whether the names are aliases for the same database.

  If you are in a partitioned database environment, when you start the governor on all partitions, the front-end utility first checks that the configuration file does not contain errors. It then reads the `db2nodes.cfg` file in the `sqllib` directory and sends a command to each database partition to start the governor front-end utility on each database partition with the start option (which, in turn, starts the daemon at each database partition).

  Note: There can only be one daemon for each logical partition for each database. As a result, it is possible to have more than one daemon on the same logical partition since you may have more than one database governed at that logical partition.

nodenum *node-num*
: Specifies the database partition on which to start the governor daemon. The number is the same as that specified in the `db2nodes.cfg` file in the `sqllib` directory.

  When you start the governor on a single database partition, the front-end utility creates a daemon to validate the governor configuration file. The governor daemon ensures that another daemon is not already running on that partition.

config-file
: Specifies the configuration file to use when monitoring the database.

  The default location for the configuration file is the `sqllib` directory. If the specified file cannot be found in the default directory, the front-end assumes that the specified name is a fully specified path name, or that the file exists in the current directory.

log-file

Specifies the base name of the file to which the governor writes log records. The log file is stored in the `log` subdirectory of the `sqllib` directory. The number of the database partition on which the governor is running is automatically appended to the log file name (for example, `mylog.0`, `mylog.1`, `mylog.2`).

stop *database*

Stops the governor daemon that is monitoring the specified database.

If you are in a partitioned database environment, the front-end utility stops the governor on all database partitions by reading the `db2nodes.cfg` file in the `sqllib` directory, and then sending a command to each database partition to call the governor front-end utility with the stop parameter. This stops the daemon at each database partition.

nodenum *node-num*

Specifies the database partition on which to stop the governor daemon. The number is the same as that specified in the `db2nodes.cfg` file in the `sqllib` directory.

When the front-end utility stops the governor daemon on a single database partition, it communicates with the daemon on that database partition by creating, moving, or deleting files in the `tmp` subdirectory of the `sqllib` directory.

The Governor Daemon

When the governor daemon is started (either by the `db2gov` front-end utility or by waking up), it runs in a loop. The first task it does is to check whether its governor configuration file has changed or has not yet been read. If either condition is true, the daemon reads the rules in the file. This allows you to change the behavior of the governor daemon while it is running.

After this, the governor daemon issues a snapshot request to obtain statistics for each application and agent working on the database.

Note: On some platforms, the CPU statistics are not available from the DB2 Monitor. Where this is the case, the account rule and the CPU limit will not be available.

The governor then checks the statistics for each application against the rules in the governor configuration file. If a rule applies to an application, the governor can: force the application; change the application's priority which indirectly changes all the agent priorities working for it on that node; or, change the schedule for the application which indirectly changes the agent priorities working on the application, depending on the action specified by the rule. The governor writes a record of any action it takes to a log file.

Note: The governor cannot be used as an alternate means to adjust agent priorities if the *agentpri* database manager configuration parameter is anything other than the system default.

When the governor finishes checking all of the applications, it sleeps for the interval specified in the configuration file. Once this time has elapsed, the governor wakes up and begins the execution loop again.

When the governor encounters an error or stop signal, it does clean-up processing before ending. The clean-up processing resets all application agent priorities (using a list of applications whose priorities have been set). It then resets the priorities of any agents that are no longer working on an application. This ensures that agents do not remain running with nondefault priorities after the governor ends. If an error occurs, a message is written to the `db2diag.log` file to indicate that the governor ended abnormally.

Note: The governor daemon is not a database application, and, therefore, does not maintain a connection to the database. (It does have an instance attachment, however.) The governor daemon can detect when the database manager ends because it can issue snapshot requests.

Creating the Governor Configuration File

When you start the governor, you specify the name of the configuration file that contains the rules to be used to govern applications running against the database. The governor acts based on these rules.

If your requirements for governing the database change, you can edit the configuration file without stopping the governor. Each governor daemon will detect that the file has changed, and reread it.

You must create the configuration file in a directory that is mounted across all the database nodes, because the governor daemon on each node must be able to read the same configuration file.

The configuration file consists of rules and comments. Most entries can be specified in uppercase, lowercase, or mixed case characters. The exception is `applname` which is case sensitive.

You delimit comments within the { } braces. The rules include:

- The database to which the rules apply.
- The length of time the governor sleeps before waking up to check the applications.
- The rules that specify how to govern the applications. These rules are made of smaller components called rule clauses.

Each rule in the file must be followed by a semicolon (;).

The following rules specify the database being monitored, and the interval at which the daemon wakes up after working through its loop of activities (which are described in “The Governor Daemon” on page 425). Each of these rules are only specified once in the file.

dbname

The name or alias of the database to be monitored.

account *nnn*

Account records are written containing CPU usage statistics for each connection at the specified number of minutes.

Note: This option is not available on OS/2.

interval

The interval, in seconds, at which the daemon wakes up. If no interval is specified, an interval of 120 seconds is used.

You combine the following rule clauses to form a rule (that is, the full rule is followed by a semicolon, and not each individual clause). The clauses specify the time during which the rule applies, the limit on resource that can be used, and, optionally, specific users or applications and any action for the governor to take if a limit specified in the rule is exceeded. The clauses can only be specified once in a rule, but can be specified in more than one rule. The clauses must be specified in the order shown. In the description that follows, a [ ] indicates an optional clause.

[desc]

Specifies a text description for the rule. The description must be enclosed by either single or double quotation marks.

[time]

Specifies the time period during which the rule is to be evaluated.

The time period must be specified in the following format `time hh:mm hh:mm`, for example, `time 8:00 18:00`. If this clause is not specified, the rule is valid 24 hours a day.

[authid]

Specifies one or more authorization ids (authid) under which the application is executing. Multiple authids must be separated by a comma (,), for example `authid gene, michael, james`. If this clause does not appear in a rule, the rule applies to all authids.

[applname]

Specifies the name of the executable (or object file) that makes the connection to the database.

Multiple application names must be separated by a comma (,), for example, `applname db2bp, batch, geneprog`. If this clause does not appear in a rule, the rule applies to all application names.

Notes:

1. Application names are case sensitive.
2. The database manager truncates all application names to 20 characters. You should ensure that the application you want to govern is uniquely identified by the first 20 characters of its application name; otherwise, an unintended application may be governed.

   Application names specified in the governor configuration file are truncated to 20 characters to match their internal representation.

setlimit

Specifies one or more limits for the governor to check. The limits can only be -1 or greater than 0 (for example, `cpu -1 locks 1000 rowssel 10000`). At least one of the limits (cpu, locks, rowsread, uowtime) must be specified. The governor can check the following limits:

cpu *nnn*

Specifies the number of CPU seconds that can be consumed by an application. If you specify `-1`, the governor does not limit the application's CPU usage.

Note: This option is not available on OS/2.

locks *nnn*

Specifies the number of locks that an application can hold. If you specify `-1`, the governor does not limit the number of locks held by the application.

rowssel *nnn*

Specifies the number of rows that are returned to the application. This value will only be non-zero at the coordinator node. If you specify `-1`, the governor does not limit the number of rows that can be selected.

uowtime *nnn*

Specifies the number of seconds that can elapse from the time that a unit of work (UOW) first becomes active. If you specify `-1`, the elapsed time is not limited.

Note: If you used the `sqlmon` (Database System Monitor Switch) API to deactivate the unit of work switch, this will affect the ability of the governor to govern applications based on the unit of work elapsed time.

idle *nnn*

Specifies the number of idle seconds allowed for a connection before a specified action is taken. If you specify `-1`, the connection's idle time is not limited.

rowsread *nnn*

Specifies the number of rows an application can select. If you specify `-1`, there is no limit on the number of rows the application can select.

Note: This limit is not the same as rowssel. The difference is that rowsread is the count of the number of rows that had to be read in order to return the result set. The number of rows read includes reads of the catalog tables by the engine and may be diminished when indices are used.

[action]

Specifies the action to take if one or more of the specified limits is exceeded. You can specify the following:

priority *nnn*

Specifies to change the priority of agents working for the application. Valid values vary depending on your operating system.

For example, valid priority range on Windows NT is from -6 to +6. See "Priority of Agents (agentpri)" on page 512 for more information on priority ranges and how to use them.

For this parameter to be effective:

- On AIX, the *agentpri* database manager parameter must be set to the default value; otherwise, it overrides the priority clause.
- On OS/2, the *agentpri* database manager parameter and *priority* action may be used together.

force

Specifies to force the agent that is servicing the application. (Issues a FORCE APPLICATION to terminate the coordinator agent.)

schedule [class]

Scheduling improves the priorities of the agents working on the applications with the goal of minimizing the average response times while maintaining fairness across all applications.

The governor enforces its schedule by setting priorities for the agents working on the applications, using query cost estimates from the DB2 internal query compiler. If the class option is specified, all applications chosen by the rule are scheduled among themselves only. If this option is not specified, the governor uses one or more classes, with scheduling done within each class.

Within each class, how an application is prioritized is based on:

- The number of locks held by the application within the class. (An application holding up many other applications due to locking is given a high priority.)

- The application's age. (An application in the system for a long time is given a high priority.)
- The application's estimated remaining running time. (An application close to finishing is given a high priority.)

Applications that are not covered by any schedule run with the highest authority.

Note: If you used the `sqlmon` (Database System Monitor Switch) API to deactivate the statement switch, this will affect the ability of the governor to govern applications based on the statement elapsed time.

Note: If a limit is exceeded and the action clause is not specified, the governor reduces the priority of agents working for the application.

If more than one rule applies to an application, the rule that is closest to the end of the configuration file is applied to the application. An exception occurs if -1 is specified for a clause in a rule. In this situation, the value specified for the clause in the subsequent rule can only override the value previously specified for the *same* clause: other clauses in the previous rule are still operative. For example, one rule indicates that the priority of an application is to be decreased if its elapsed time is greater than 1 hour, or if it selects more than 100 000 rows (that is, `rowssel 100000 uowtime 3600`). A subsequent rule indicates that the same application can have unlimited elapsed time (that is, `uowtime -1`). In this situation, if the application runs for more than 1 hour, its priority won't be changed (that is, `uowtime -1` overrides `uowtime 3600`), but if it selects more than 100 000 rows, its priority will be lowered (as `rowssel 100000` is still valid).

Figure 54 on page 431 shows an example of a configuration file.

```
{ Wake up once a second, the database name is ibmsampl
 do accounting every 30 minutes.  }
interval 1; dbname ibmsampl; account 30;

desc "CPU restrictions apply 24 hours a day to everyone"
setlimit cpu 600 rowssel 1000000 rowsread 5000000;

desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600;

desc 'Slow down a subset of applications'
applname jointA, jointB, jointC, quryA
setlimit cpu 3 locks 1000 rowssel 500 rowsread 5000;

desc "Have governor prioritize these 6 long apps in 1 class"
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
action schedule class;

desc "Schedule all applications run by the planning dept"
authid planid1, planid2, planid3, planid4, planid5
setlimit cpu -1
action schedule;

desc "Schedule all CPU hogs in one class which will control consumption"
setlimit cpu 3600
action schedule class;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp
setlimit cpu 5 locks 100 rowssel 250;

desc "During day hours do not let anyone run for more than 10 seconds"
time 8:30 17:00 setlimit cpu 10 action force;

desc "Allow users doing performance tuning to run some of
     their applications during lunch hour"
time 12:00 13:00 authid ming, geoffrey, john, bill
applname tpcc1, tpcc2, tpcA, tpvG setlimit cpu 600 rowssel 120000 action force;

desc Some people should not be limited -- database administrator
  and a few others.  As this is the last specification in the
  file, it will override what came before."
authid gene, hershel, janet setlimit cpu -1 locks -1 rowssel -1;

desc "Increase the priority of an important application so it always
      completes quickly"
applname V1app setlimit cpu 1 locks 1 rowssel 1 action priority -20;
```

Figure 54. Example Governor Configuration File

Governor Log Files

When a governor daemon forces an application, reads the governor configuration file, changes an application's priority, encounters an error or warning, starts, or ends, it writes a record to a log file. A separate log file exists for each governor daemon. This prevents file-locking bottlenecks that would result from many governor daemons writing to the same file at the same time. You can use the `db2govlg` utility to merge the log files together and query them. This utility is described in "Querying Governor Log Files" on page 433.

The log files are stored in the `log` subdirectory of the `sqllib` directory. You provide the base name for the log file when you issue the `db2gov` command. You should ensure that the log file name contains the database name, because there will be a log file for each node of each database that is being governed. In a partitioned database environment, the node number of the database partition that the governor is running on is automatically appended to the log file name to ensure that the filename is unique for each governor.

Each record in the log file has the following format:

```
Date Time NodeNum RecType Message
```

The *Date* and *Time* field is in the yyyy-mm-dd-hh.mm.ss format, so that you can merge the log files for each database partition by sorting on this field.

The *NodeNum* field indicates the number of the database partition on which the governor is running.

The *RecType* field contains different values, depending on the type of log record being written to the log. The values that can be recorded are:

- `START` to indicate that the governor was started
- `FORCE` to indicate that an application was forced
- `PRIORITY` to indicate that the priority of an application was changed
- `ERROR` to indicate an error
- `WARNING` to indicate a warning
- `READCFG` to indicate that the governor read the configuration file
- `STOP` to indicate that the governor was stopped
- `ACCOUNT` to indicate the application's accounting statistics.

  The fields are:

  - `authid`
  - `appl_id`
  - `written_usr_cpu`
  - `written_sys_cpu`

– appl_con_time

- SCHEDULE to indicate that a change in agent priorities occurred.

Because standard values are written, you can query the log files for different types of actions. The *Message* field provides other nonstandard information that varies according to the value under the *Rectype* field. For instance, a FORCE or NICE record indicates application information in the *Message* field, while an ERROR record includes an error message.

An example log file is as follows:

```
1995-12-11 14.54.52    0 START      Database = TQTEST
1995-12-11 14.54.52    0 READCFG    Config = /u/db2instance/sqllib/tqtest.cfg
1995-12-11 14.54.53    0 ERROR      SQLMON Error: SQLCode = -1032
1995-12-11 14.54.54    0 ERROR      SQLMONSZ Error: SQLCode = -1032
```

Querying Governor Log Files

Each governor daemon writes to its own log file. You can use db2govlg utility to query the log file. You can list the log files for a single partition, or for all database partitions, sorted by date and time. You can also query on the basis of the *RecType* log field. The syntax for db2govlg is as follows:

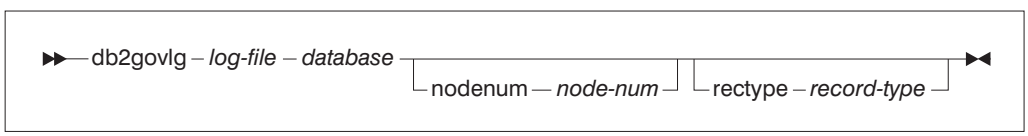


Figure 55. Syntax for db2govlg

The parameters are as follows:

log-file
: The base name of the log file (or files) that you want to query.

database
: The database that the governor is monitoring.

nodenum *node-num*
: The node number of the database partition on which the governor is running.

rectype *record-type*
: The type of record that you want to query. The record types are:

  - START
  - READCFG
  - STOP
  - FORCE

- NICE
- ERROR
- WARNING
- ACCOUNT

There are no authorization restrictions for using this utility. This allows all users to query whether the governor has affected their application. If you want to restrict access to this utility, you can change the group permissions for the `db2govlg` file.

Running the Governor and Database Manager Performance

The governor can affect database manager performance because it requests snapshots of the database manager. If the governor uses too much CPU, you can increase its wake-up interval to reduce its CPU usage.

Chapter 16. Redistributing Data Across Database Partitions

Only if you are working in a partitioned database environment do you need to be concerned with redistribution of data. If you are in a single partition database environment there is no need for you to use the information found here.

You use the Data Redistribution utility to move data among the database partitions in an existing nodegroup. You can use it to do the following:

- Balance data volumes and processing loads across database partitions.

  This is useful if you have a database table in which all the data is accessed on a regular basis.

- Introduce skew in the data distribution across database partitions.

  This is useful if you have a database table in which only some of the data is accessed on a regular basis. In this situation, you could redistribute the table so that the infrequently accessed data is on a small number of database partitions in the nodegroup, and the frequently accessed data is distributed over a larger number of partitions. This would improve access performance and throughput on the most frequently run applications.

- Add database partitions to a nodegroup. (Provided for backward compatibility only with DB2 for Parallel Edition. The recommended way to add a database partition is to use the ALTER NODEGROUP command.)

- Drop database partitions from a nodegroup. (Provided for backward compatibility only with DB2 for Parallel Edition. The recommended way to drop a database partition is to use the ALTER NODEGROUP command.)

To preserve table collocation, this operation is applied to all tables in a nodegroup, and redistribution is done at the nodegroup level rather than at the table level.

To achieve the data distribution that you want, the utility uses a partitioning map to move the rows of the tables among the database partitions of the nodegroup. Depending on the option you specify, the utility can generate a target partitioning map or can use an existing partitioning map as input.

Note: You should specify a log file size based on the log space requirements you think that the Data Redistribution operation will need. You should also ensure that the log is large enough to accommodate the INSERT and DELETE operations done at each database partition where data is being redistributed.

How to Partition Data

By default, the Data Redistribution utility assumes that the same number of rows hash to each hash partition, therefore it partitions the data uniformly across all the database partitions of the nodegroup. If the same number of rows do not hash to each partition, you can use a *distribution file* to specify the current distribution. This file contains a value for each of the 4096 hash partitions. Each value is used as the *weight* of the

corresponding partition. The Data Redistribution utility generates a target partitioning map in which all the database partitions have about the same weight. Thus, the distribution file can be used to achieve uniform data distribution even if the data distribution is skewed.

The `AutoLoader` utility can be used to create a data distribution file using the ANALYZE option. You can use this file as input to the Data Redistribution utility. For more information, see “Using the AutoLoader Utility” on page 153.

Alternatively, you can use the PARTITION and NODENUMBER SQL functions to determine the current data distribution across database partitions. You can use this information to derive both a distribution file and a target partitioning map.

Adding and Dropping Database Partitions

You can use the ALTER NODEGROUP statement to add or drop database partitions from a nodegroup. When adding database partitions, the partitions must already be defined in the `db2nodes.cfg` file.

Following the use of the ALTER NODEGROUP statement, a new partitioning map is created. This new partitioning map can become the target partitioning map when using the Data Redistribution utility. (The other way to create the target partitioning map is to create it yourself.)

Specifying a Target Partitioning Map

The Data Redistribution utility uses a partitioning map to do the data redistribution. It can create its own target partitioning map, or you can provide one for the utility to use. If you create one, the entry or entries determine the type of nodegroup that results from the data redistribution:

- 1 entry for a single-partition nodegroup
- 4096 entries for a multipartition nodegroup

If the target partitioning map has more than one database partition, all tables in the nodegroup must have a partitioning key defined.

The target partitioning map can only contain databases which are defined in the SYSCAT.NODEGROUPDEF catalog table, excluding those with an IN_USE value of 'T'. ('T' means that the partition is not in the target partitioning map.) For all partitions that have an IN_USE value of 'D' (meaning to drop), and which also do not appear in the target partitioning map, these databases are dropped when the redistribution operation has completed successfully.

How Data Is Redistributed Across Database Partitions

The Data Redistribution operation is done on the set of tables in the specified nodegroup of a database. (The application must be connected to the database at the catalog database partition before executing the operation.) The utility uses both the

source partitioning map and the target partitioning map to identify which partitions have been assigned to a new location (that is, a new node number). All rows that correspond to a partition that has a new location are moved from the database partition specified in the source partitioning map to the database partition specified in the target partitioning map.

The Data Redistribution utility does the following:

1. Obtains a new partitioning map ID for the target partitioning map, and inserts it into the SYSPARTITIONMAPS catalog table.
2. Updates the REBALANCE_PMAP_ID column in the SYSNODEGROUPS catalog table for a nodegroup with the new partitioning map ID.
3. Adds any new database partitions to the SYSNODEGROUPDEF catalog table.
4. Sets the IN_USE column in the SYSNODEGROUPDEF catalog table to D for any database partition that is to be dropped.
5. Does a COMMIT for the catalog updates.
6. Creates database files for all new database partitions.
7. Redistributes the data in each table for every table in the nodegroup. This is described in "How Data Is Redistributed in Tables."
8. Deletes database files and deletes entries in the SYSNODEGROUPDEF catalog table for database partitions that were previously marked to be dropped.
9. Updates the nodegroup record in the SYSNODEGROUPS catalog table to set PMAP_ID to the value of REBALANCE_PMAP_ID and REBALANCE_PMAP_ID to NULL.
10. Deletes the old partitioning map from the SYSPARTITIONMAPS catalog table.
11. Does a COMMIT for all changes.

How Data Is Redistributed in Tables

When doing data redistribution on a table, the utility does the following:

1. Locks the row for the table in the SYSTABLES catalog table.
2. Invalidates all plans that involve this table. The partitioning map ID associated with the table will change because the table is being redistributed. Because the plans are invalidated, the compiler must obtain the new partitioning information for the table and generate plans accordingly.
3. Locks the table in exclusive mode.
4. Redistributes the data in the table.
5. If the redistribution operation succeeds, it:
   a. Issues a COMMIT for the table.
   b. Continues with the next table in the nodegroup.

   If the operation fails before the table is fully redistributed, the utility:

a. Issues a ROLLBACK on updates to the table.

b. Ends the entire redistribution operation and returns an error.

Recovering From Redistribution Errors

If the data redistribution operation fails, some tables may be redistributed while others are not. This occurs because data redistribution is performed a table at a time. You have two options for recovery:

- Use the CONTINUE option to continue the operation to redistribute the remaining tables.
- Use the ROLLBACK option to undo the redistribution and set the redistributed tables back to their original state. The roll-back operation can take about the same amount of time as the original redistribution operation.

Before you can use either option, a previous data redistribution operation must have failed such that the REBALANCE_PMID column in the SYSNODEGROUPS catalog table is set to a non-NULL value.

After the redistribution operation begins to execute, a file is written to the `redist` sub-directory of the `sqllib` directory. This file has the following naming convention:

databasename.nodegroupname.timestamp (for UNIX platforms)
*databasename**nodegroupname**date**time* (for non-UNIX platforms)

Note: On non-UNIX platforms, only the first eight (8) bytes of the `nodegroupname` are used.

This file lists any operations that are done on nodes, the names of the tables that were redistributed, and the completion status of the operation. If a table cannot be redistributed, its name and the applicable SQLCODE is listed. If the redistribution operation cannot begin because of an incorrect input parameter, the file is not written and an SQLCODE is returned.

Should you happen to delete this file by mistake, you are still able to attempt a CONTINUE operation.

Data Redistribution and Other Operations

You can do the following operations on objects of the nodegroup while the utility is running. You cannot, however, do them on the table that is being redistributed. You can:

- Create indexes on other tables. The CREATE INDEX statement uses the partitioning map of the affected table.
- Drop other tables. The DROP TABLE statement uses the partitioning map of the affected table.
- Drop indexes on other tables. The DROP INDEX statement uses the partitioning map of the affected table.

- Query other tables.
- Update other tables.
- Create new tables a table space of the nodegroup. The CREATE TABLE statement uses the target partitioning map.
- Create table spaces in the nodegroup.

You cannot do the following operations while the utility is running:

- Another redistribution operation on the nodegroup
- An ALTER TABLE on any table in the nodegroup
- Drop the nodegroup
- Alter the nodegroup.

Following Data Redistribution

After completing the redistribution of data across a nodegroup, it is strongly recommended that you do a RUNSTATS to update the statistics associated with the tables that may have been redistributed.

For more information on the RUNSTATS command, refer to the *Command Reference* manual.

Chapter 17. Scaling Your Configuration

You may find that the size of your configuration is not appropriate for your needs. You may have tried increasing your configuration memory, or storage capacity, or both, but this has not provided you with sufficient improvement to meet your current or future needs.

You should consider scaling your configuration as discussed in the remainder of this chapter if:

- You had a single-partition configuration with a single processor that was being used to its maximum capacity. As a result, you have decided to change configurations and have:
  - Determined a symmetric multiprocessor (SMP) configuration is your best choice for a new environment. You perhaps made this choice because you want to take advantage of the processing power available with more than one processor. Each processor shares memory and storage system resources. All of the processors are within one system, so there are no additional considerations such as communication lines between systems, perhaps no additional administration staff to support any new systems, and coordination of tasks between systems is not an issue. DB2 Universal Database supports this environment.
  - Determined a partitioned database configuration is your best choice for a new environment. You perhaps made this choice because you want to take advantage of the processing power available with more than one processor that is physically separate from the first. Each processor has it's own memory and storage system resources without having to share with the other processor. While you may have the additional considerations mentioned above (communications, staff, and coordination of tasks), there are advantages to this choice such as the ability to balance data and user access across more than one system. DB2 Universal Database supports this environment.
- You currently have a SMP configuration and you are planning to add one or more additional processors. In this case, you are already familiar with those considerations associated with this type of environment. By adding one or more additional processors, you are simply adding complexity to your environment without adding new considerations. DB2 Universal Database supports this environment.
- You have a partitioned database configuration and you are planning to add one or more additional database partitions. In this case, you are already familiar with those considerations associated with this type of environment. By adding one or more additional database partitions, you are simply adding complexity to your environment without adding new considerations. DB2 Universal Database supports this environment.
- You have a partitioned database configuration and you are planning to add one or more additional database partitions each of which may be in a SMP configuration. DB2 Universal Database supports this environment.

When you scale your system by changing the environment, you should be aware of the impact that such a change can have on your database procedures such as backing up and restoring the database.

When you add a new database partition, you cannot drop or create a database until the procedure is complete, and the new server is successfully integrated into the system.

Adding Processors to a Machine

The first thing to be done, of course, is to ensure that you have installed one or more additional processors in your machine. To allow the DB2 database manager to take advantage of the new processors, there are configuration parameters that should be reviewed and perhaps updated. (Some operating systems, like Solaris, can dynamically vary processors on- and off-line.) The parameters that are used to determine the number of processors used and may need to be updated include:

- "Enable Intra-Partition Parallelism (intra_parallel)" on page 560
- "Default Degree (dft_degree)" on page 542
- "Maximum Query Degree of Parallelism (max_querydegree)" on page 559

You should also consider the parameters associated with applications that may need to be updated. Refer to "Parallel Processing of Applications" on page 298 for more information.

Utilities in DB2 such as LOAD, backup, and restore can take advantage of the additional processors. Refer to Chapter 5, "Utilities for Moving Data" on page 141 and Chapter 6, " Recovering a Database" on page 179 for information on these utilities.

Adding Database Partitions to a System

You can add database partitions to the system either when it is running, or when it is stopped. The following sections describe how to do this task. Because adding a new server can be time consuming, you should do it when the database manager is already running. The procedure is described in "Adding Database Partitions to a Running System" on page 443.

The ADD NODE command is used to add a database partition to a system. This command can be invoked:

- As an option on `db2start`
- Using:
  - The command line processor ADD NODE command
  - The `sqleaddn_api`
  - The `sqlepstart_api`.

The method you use to invoke the command is dependent upon whether your system is stopped (using `db2start`) or running (using any of the other choices).

When a new database partition is added to the system using the ADD NODE command, all existing databases in the system are created on the new database partition. You can also specify which containers for temporary table spaces will be used with the databases that are created. The containers can be:

- The same as those defined for the catalog node for each database. (This is the default.)
- The same as those defined for another partition.
- Not created at all. The ALTER TABLESPACE statement must be used to add temporary table space containers to each database before the database can be used.

A database on the new partition cannot be used to contain data until the nodegroups are altered to include the new database partition. See "Adding and Dropping Database Partitions" on page 436 for more information on how to alter a nodegroup.

Note: If there are no databases defined in the system, edit the `db2nodes.cfg` file to add a new database partition definition; do not use any of the following procedures, as an error will result. See "Altering a Nodegroup" on page 99 for more information on how to update the node configuration file.

Adding Database Partitions to a Running System

You can add new database partitions to a multiple server system while it is running and while applications are connected to databases. However, a newly added server does not become available to all databases until the database manager is shut down and restarted.

To add a database partition to a multiple server system:

1. If the database partition is to be created on a server that already exists in the system, go to the next step. Otherwise, do the following:
   a. Install the new server. This includes making executables accessible (using shared file-system mounts or local copies), synchronizing operating system files with those on existing processors, ensuring that the `sqllib` directory is accessible as a shared file system, and ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values.
   b. Register the host name with the name server in the `hosts` file in the `etc` directory on all database partitions.
2. Run the DB2START command on any database partition, specifying the NODENUM, ADDNODE, HOSTNAME, PORT, and NETNAME parameters. The values that you specify for these parameters are used to update the `db2nodes.cfg` file. You can also optionally specify the source for any temporary table spaces which need to be created with the databases. If no table space information is provided, the temporary table space definitions are retrieved from the catalog node for each database.

When the command completes, the new server is stopped. The db2nodes.cfg file is not updated with the new server information until DB2STOP is executed. This ensures that the Add Node utility (which is called when the ADDNODE parameter is specified) runs on the correct database partition. When the utility ends, the new server is stopped.

3. Stop the database manager by running the DB2STOP command.

   When you stop all the nodes in the system, the `db2nodes.cfg` file is updated to include the new database partition.

4. Start the database manager by running the DB2START command.

   The newly added database partition is now started along with the rest of the system.

   When all the database partitions in the system are running, system-wide activities, such as creating or dropping a database, can be done.

5. Optionally, take a backup of all databases on the new database partition.
6. Optionally, redistribute data to the new database partition. For details, see Chapter 16, “Redistributing Data Across Database Partitions” on page 435.

Adding Database Partitions to a Stopped System

You can add new database partition to a multiple server system while it is stopped. The newly added server becomes available to all databases when the database manager is started up again. You have two options. You can either have the database manager update the `db2nodes.cfg` file for you, or you can do it manually. The preliminary steps for both procedures are the same.

To add a new database partition to a multiple server system:

1. Issue DB2STOP to stop all the servers.
2. If the server is to be created on a processor that already exists in the system, go to the next step. Otherwise, do the following:
   a. Install the new processor. This includes making executables accessible (using shared file-system mounts or local copies), synchronizing operating system files with those on existing processors, ensuring that `sqllib` directory is accessible as a shared file system, and ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values.
   b. Register the host name with the name server in the `hosts` file in the `etc` directory on all database partitions.
   c. If you want the database manager to update the `db2nodes.cfg` file for you, continue with the instructions in “Having the Database Manager Update db2nodes.cfg” on page 445.

      If you want to update the `db2nodes.cfg` file yourself, continue with the instructions in “Updating db2nodes.cfg Manually” on page 445.

Having the Database Manager Update db2nodes.cfg

Continue the procedure as follows:

1. Run the DB2START command on the new database partition specifying NODENUM, ADDNODE, HOSTNAME, PORT, and NETNAME parameters. The values that you specify for these parameters are used to update the `db2nodes.cfg` file.

   When the command completes, the new server is stopped. The db2nodes.cfg file is not updated with the new server information until DB2STOP is executed. This ensures that the Add Node utility (which is called when the ADDNODE parameter is specified) runs on the correct database partition. When the utility ends, the new server is stopped.

2. Issue the DB2STOP command.

   When you issue the DB2STOP command, the `db2nodes.cfg` file is updated to include the new server.

3. Issue the DB2START command to start the database system.
4. Optionally, take a backup of all databases on the new database partition.
5. Optionally, redistribute data to the new server. For details, see Chapter 16, “Redistributing Data Across Database Partitions” on page 435.

Updating db2nodes.cfg Manually

Continue the procedure as follows:

1. Edit the `db2nodes.cfg` file and add the new database partition to it.
2. Issue the following command to start the new node:

   ```
   DB2START NODENUM nodenum
   ```

   Specify the number you are assigning to the new database partitioned server as the value of *nodenum*.

3. If the new server is to be a logical database partition (that is, it is not node 0), use **db2set** command to update the DB2NODE registry value, specifying the number of the server you are adding.
4. Run the Add Node utility on the new server.

   This utility also causes a database partition to be created locally for every database that already exists in the system. The database parameters for the new partitions are set to the default value, and each partition remains empty until you move data to it.

5. When the Add Node utility completes, issue the DB2START command to start the other database partitions in the system.

   You should not attempt to do any system-wide activities, such as creating or dropping a database, until all servers are successfully started.

6. Optionally, take a backup of all databases on the new server.

7. Optionally, redistribute data to the new database partition. For details, see Chapter 16, “Redistributing Data Across Database Partitions” on page 435.

Dropping a Database Partition from a System

You can drop a database partition by using the DB2STOP command with the DROP NODENUM parameter, or the `sqlepstp` API. Before doing this, you must first ensure that the server being dropped is not being used by any database. To check, issue the DROP NODE VERIFY command.

You should ensure that all transactions for which this database partition was the coordinator have all committed or rolled back successfully. This may require doing crash recovery on other servers.

For example, if you drop the coordinator database partition and another server participating in a transaction crashed before the coordinator server was dropped, the crashed server will not be able to query the coordinator server for the outcome of any indoubt transactions.

To drop a database partition from a multiple server system:

1. Redistribute the data for every database that resides on this node. This ensures that the partitioning map is kept current. For details, see Chapter 16, “Redistributing Data Across Database Partitions” on page 435.
2. Issue the DROP NODE VERIFY command or the `sqledrpn` API to verify that the server is not in use.

Depending on the message you receive, proceed with either step 3 or step 4.

3. If you receive message SQL6034W (Node not used in any database), you can do the following:
   a. Issue the DB2STOP command with the DROP NODENUM parameter to drop the database partition. After the command completes successfully, the system is stopped.
   b. If you want to, start the database manager with the DB2START command.
4. If you receive message SQL6035W (Node in use by database), do the following:
   a. Use the REDISTRIBUTE NODEGROUP command to redistribute the data from the database partition you are deleting to other servers from the database alias, as indicated in message SQL6035W. You cannot drop the server until this is done.
   b. Drop any event monitors defined on the database partition.
   c. Return to step 2 and continue.

Chapter 18. Benchmark Testing

Benchmarking is a normal part of the application development life cycle. It is a team effort involving both application developers and database administrators (DBAs), and should be performed against your application in order to determine and improve performance. Assuming that the application code has been written as efficiently as possible, additional performance gains can be realized from tuning the database and database manager configuration parameters to meet the requirements of the application.

There are several different types of benchmarking. A *transaction per second* benchmark would determine the throughput capabilities of the database manager under certain limited laboratory conditions. An *application* benchmark would test the same throughput capabilities, but under conditions that are closer to those under which your application will run when it is implemented. Benchmarking for the purpose of tuning configuration parameters is based upon these “real-world” conditions, and involves repeatedly running SQL taken from your application with varying parameter values until your application runs as efficiently as possible.

The benchmarking methods described in this section are oriented towards the configuration parameters. However, the same basic technique can be used for tuning other factors that affect performance, such as:

- SQL statements
- Indexes
- Table space configuration
- Application code
- Hardware configuration.

Benchmarking is helpful in understanding how the database manager responds under varying conditions. You could create scenarios that test deadlock handling, utility performance, different methods of loading data, transaction rate characteristics as more users are added, and even the effect on the application of using a new release of the product.

The following topics are provided:

Benchmark Testing Methodology

This benchmarking technique is based on the scientific method. A repeatable environment will be created in which the same test, run under the same conditions, will yield comparable results.

Benchmarking can also begin by running the test application in a normal environment. As a performance problem is narrowed down, specialized test cases can be developed to limit the scope of the function that is being tested and observed. The specialized test cases need not emulate an entire application in order to obtain valuable information. Start with simple measurements, and increase the complexity only when warranted.

Characteristics of good benchmarks (or measurements) include:

- Each test is repeatable.
- Each iteration of a test is started in the same system state.
- There are no functions or applications active in the system other than those being measured (unless the scenario includes some amount of other activity going on in the system).

  Note: Applications that are started use memory even when they are minimized or idle. This increases the likelihood of paging skewing the results of the benchmark and violating the repeatability rule.

- The hardware and software used for benchmarking matches your production environment.

As with any benchmarking, a scenario must be devised and then executed. The following information applies these concepts to the DB2 environment.

- "Preparing for Benchmark Testing"
- "Creating a Benchmark Program" on page 450
- "Executing the Benchmark Tests" on page 455.

Preparing for Benchmark Testing

The logical design of your application's database should be complete before performance benchmarking is started. Tables, views, and indexes need to be set up and populated. Tables should be normalized, application packages bound, and tables populated with realistic data.

You should have determined the final physical design of the database. The database manager objects should be placed in their final disk locations, log files sized, work files and backup locations determined, and backup procedures tested. In addition, packages should be checked to make sure that performance options such as row blocking are enabled when possible.

You should have reached a point in the application's programming and testing phases that will enable you to create your benchmark programs (see next section). An application's practical limits may be revealed during the benchmark testing; however, the purpose of the benchmark described here is to measure performance, not to detect defects or abends.

Your benchmarking test program will need to run in as accurate a representation of the final production environment as possible; ideally, on the same model of server with the same memory and disk configurations. This is especially important when the application will ultimately involve large numbers of users and large amounts of data. The operating

system itself and any communications or file-serving facilities used directly by the benchmark should also have been tuned.

It is also important to benchmark with a production-size database. An individual SQL statement should return as much data and involve as much sorting as it will once it is implemented in production. Adhering to this rule will ensure that the application will incur representative memory requirements.

The type of SQL statements to be benchmarked should be either *representative* or *worst-case*, as described below:

Representative SQL

Representative SQL includes those statements that are executed during typical operations of the application being benchmarked. The statements that are selected will depend on the nature of the application. For example, a data-entry application might test an INSERT statement, while a banking transaction might test a FETCH, an UPDATE, and several INSERTs. The frequency of execution and volume of data processed by the statements chosen should be considered average. If the volumes are excessive, the statements should be considered under the *worst-case* category, even if they are typical SQL statements.

Worst-case SQL

Statements falling in this category include:

- Statements that are executed frequently.
- Statements that have high volumes of data being processed.
- Statements that are time-critical.

  For example, an application that is run when a telephone call is received from a customer and the statements must be run to retrieve and update the customer's information while the customer is waiting.
- Statements with the largest number of tables being joined or with the most complex SQL in the application.

  For example, a banking application that produces combined customer statements of monthly activity for all their different types of accounts. A common table may list customer address and account numbers; however, several other tables must be joined to process and integrate all of the necessary account transaction information. Multiply the work necessary for one account by the several thousand accounts that must be processed during the same period, and the potential time savings drives the performance requirements.
- Statements that have a poor access path, such as one that is not executed very often and is not supported by the indexes that have been created for the table(s) involved.
- Statements that have a long elapsed time.
- A statement that is only executed at application initialization but has disproportionate resource requirements.

  For example, an application that generates a list of account work that must be processed during the day. When the application is started, the first major SQL statement causes a 7-way join, which creates a very large list of all the

accounts for which this application user is responsible. The statement might only be run a few times per day, but takes several minutes to run when it has not been tuned properly.

Creating a Benchmark Program

There are a variety of factors to consider when designing and implementing a benchmark program. Since the main purpose of the program is to simulate a user application, the overall structure of the program can vary. You can use the entire application as the benchmark and simply introduce a means for timing the SQL statements to be analyzed. For large or complex applications, it may be more practical to just include blocks containing the important statements.

To test the performance of specific SQL statements, another approach would be to include these statements alone in the benchmark program along with the necessary CONNECT, PREPARE, OPEN, and other statements and a timing mechanism.

Another factor to consider is the type of benchmark to use. One option is to run a set of SQL statements repeatedly over a time interval. The ratio of the number of statements executed and this time interval would give the throughput for the application. Another option would be to simply determine the time required to execute individual SQL statements.

Regardless of the type of benchmark program, an efficient timing system is necessary to calculate the elapsed time, whether for individual SQL statements or the application as a whole. For simulating applications in which individual SQL statements would be executed in isolation, it may be important to consider times for CONNECT, PREPARE, and COMMIT statements. However, for programs processing many different statements, perhaps only a single CONNECT or COMMIT is necessary, so focusing on just the execution time for an individual statement may be the priority.

While the elapsed time for each query is an important factor in performance analysis, it may not necessarily reveal bottlenecks. For example, information on CPU usage, locking, and buffer pool I/O could show that the application is I/O bound instead of using the CPU to its full capacity. A benchmark program should allow you to obtain this kind of data for a more detailed analysis if needed.

Not all applications will need to send the entire set of rows retrieved from a query to some output device. For example, some may use the whole answer set as input for another program (that is, none of the rows are sent to output). Formatting data for screen output usually has high CPU cost and may not reflect user need. In order to provide an accurate simulation, a benchmark program should reflect the row handling of the specific application. If rows do get sent to an output device, inefficient formatting could consume the majority of CPU processing time and misrepresent the actual performance of the SQL statement itself.

The db2batch Benchmark Tool: A benchmark tool (`db2batch`) is provided in the `misc` subdirectory of your instance `sqllib` directory. This tool takes many of the points made above regarding the creating of a benchmark program into consideration. This tool will

read SQL statements from either a flat file or standard input, dynamically describe and prepare the statements, and return an answer set. It also provides the added flexibility of allowing you to control the size of the answer set, as well as the number of rows that should be sent from this answer set to an output device.

You can also specify the level of performance-related information supplied, including the elapsed time, CPU and buffer pool usage, locking, and other statistics collected from the database monitor. If you are timing a set of SQL statements, `db2batch` will also summarize the performance results and provide both arithmetic and geometric means. For more information on invocation syntax, and options, type `db2batch -h` on a command line.

The *Command Reference* manual can also be referenced for more information on `db2batch`.

The following is an example of how `db2batch` could be used with an input file `db2batch.sql`:

```
-- db2batch.sql
-- ------------
--#SET PERF_DETAIL 3 ROWS_OUT 5

-- This query lists employees, the name of their department
-- and the number of activities to which they are assigned for
-- employees who are assigned to more than one activity less than
-- full-time.
--#COMMENT Query 1
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2;
--#SET PERF_DETAIL 1 ROWS_OUT 5
--#COMMENT Query 2
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2;
```

Figure 56. Sample Benchmark Input File: db2batch.sql

Using the following invocation of the benchmark tool:

```
db2batch -d sample -f db2batch.sql
```

Produces the following output:

```
--#SET PERF_DETAIL 3 ROWS_OUT 5
Query 1

Statement number: 1

select lastname, firstnme,
deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2
```

Figure 57. Sample Output From db2batch (Part 1)

```
LASTNAME         FIRSTNME      DEPTNAME                        NUM_ACT
--------------------------------------------------------------------------
JEFFERSON        JAMES         ADMINISTRATION SYSTEMS                3
JOHNSON          SYBIL         ADMINISTRATION SYSTEMS                4
NICHOLLS         HEATHER       INFORMATION CENTER                    4
PEREZ            MARIA         ADMINISTRATION SYSTEMS                4
SMITH            DANIEL        ADMINISTRATION SYSTEMS                7
Number of rows retrieved is:        5
Number of rows sent to output is:   5
Elapsed Time is:           0.074      seconds
Locks held currently                        = 0
Lock escalations                            = 0
Total sorts                                 = 5
Total sort time (ms)                        = 0
Sort overflows                              = 0
Buffer pool data logical reads              = 13
Buffer pool data physical reads             = 5
Buffer pool data writes                     = 0
Buffer pool index logical reads             = 3
Buffer pool index physical reads            = 0
Buffer pool index writes                    = 0
Total buffer pool read time (ms)            = 23
Total buffer pool write time (ms)           = 0
Asynchronous pool data page reads           = 0
Asynchronous pool data page writes          = 0
Asynchronous pool index page reads          = 0
Asynchronous pool index page writes         = 0
Total elapsed asynchronous read time        = 0
Total elapsed asynchronous write time       = 0
Asynchronous read requests                  = 0
LSN Gap cleaner triggers                    = 0
Dirty page steal cleaner triggers           = 0
Dirty page threshold cleaner triggers       = 0
Direct reads                                = 8
Direct writes                               = 0
Direct read requests                        = 4
Direct write requests                       = 0
Direct read elapsed time (ms)               = 0
Direct write elapsed time (ms)              = 0
Rows selected                               = 5
Log pages read                              = 0
Log pages written                           = 0
Catalog cache lookups                       = 3
Catalog cache inserts                       = 3
Buffer pool data pages copied to ext storage    = 0
Buffer pool index pages copied to ext storage   = 0
Buffer pool data pages copied from ext storage  = 0
Buffer pool index pages copied from ext storage = 0
Total Agent CPU Time (seconds)              = 0.02
Post threshold sorts                        = 0
Piped sorts requested                       = 5
Piped sorts accepted                        = 5
```

Figure 58. Sample Output From db2batch (Part 1)

```
--#SET PERF_DETAIL 1 ROWS_OUT 5
Query 2
Statement number: 2
select lastname, firstnme,
deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2
LASTNAME         FIRSTNME      DEPTNAME                       NUM_ACT
------------------------------------------------------------------------
GEYER            JOHN          SUPPORT SERVICES                     2
GOUNOT           JASON         SOFTWARE SUPPORT                     2
HAAS             CHRISTINE     SPIFFY COMPUTER SERVICE DIV.         2
JONES            WILLIAM       MANUFACTURING SYSTEMS                2
KWAN             SALLY         INFORMATION CENTER                   2
Number of rows retrieved is:        8
Number of rows sent to output is:   5
Elapsed Time is:           0.037      seconds
Summary of Results
==================
                Elapsed             Agent CPU         Rows      Rows
Statement #     Time (s)            Time (s)          Fetched   Printed
1                    0.074               0.020         5         5
2                    0.037       Not Collected         8         5
Arith. mean     0.055
Geom.  mean     0.052
```

Figure 59. Sample Output from db2batch (Part 2)

The above sample output includes specific data elements returned by the database system monitor. For more information about these and other monitor elements, see the *System Monitor Guide and Reference* manual.

In the next example, just the summary table is produced.

```
db2batch -d sample -f db2batch.sql -r /dev/null,
```

Produces just the summary table. Using the `-r` option, `outfile1` was replaced by `/dev/null` and `outfile2` (which contains just the summary table) is empty, so `db2batch` sends the output to the screen:

```
Summary of Results
==================
                Elapsed             Agent CPU         Rows      Rows
Statement #     Time (s)              Time (s)        Fetched   Printed
1                     0.074               0.020         5         5
2                     0.037       Not Collected         8         5
Arith. mean     0.055
Geom.  mean     0.052
```

Figure 60. Sample Output from db2batch -- Summary Table Only

This benchmarking tool also has a CLI option. With this option, you can specify a cache size. In the following example, `db2batch` is run in CLI mode with a cache size of 30 statements:

```
db2batch -d sample -f db2batch.sql -cli 30
```

Executing the Benchmark Tests

One type of database benchmark involves choosing a configuration parameter and running the test with different values for that parameter until the maximum benefit is achieved. A single test should include executing the application through several iterations (for example, 10 times) with the same parameter value to get an average timing, which will better show the effect of parameter changes.

When running your benchmark, the first iteration should be considered a separate case from the subsequent iterations. This is because the results from the first iteration will include some start-up activities (such as initializing the buffer pool). Consequently, this iteration will take somewhat longer than the others. Although the information from this iteration *may be* realistically valid, it will not be statistically valid. Therefore, when calculating the average timing for a specific set of parameter values, use the timings from the second and subsequent iterations.

You may want to consider using the Configuration SmartGuide to create the first iteration of the benchmark. The questions asked as part of the Configuration SmartGuide will provide insight into some of those things to consider when adjusting the configuration of your environment for subsequent iterations during your benchmark activity. To use the Configuration SmartGuide, enter `db2cc` to get into the Command Center and proceed from there.

If you are benchmarking using individual queries, you need to ensure that you minimize the potential effects of previous queries. This can be accomplished by flushing the buffer pool which can be done by reading a number of pages (irrelevant to your query) to fill the buffer pool.

After completing the iterations for a single set of parameter values, a single parameter can be changed. However, between each iteration, the following tasks should be performed to restore the benchmark environment to its original state:

- Return the application data and database manager statistics to their original state. If the catalog statistics were updated for the test, ensure the same values for the statistics are used for every iteration. The data used in the tests must be consistent if it is updated in the course of the tests. This can be done by:
  - Using the RESTORE utility to restore the entire database. The backup copy of the database would be in its previous state, and ready for the next test.
  - Using the IMPORT or LOAD utility to restore an exported copy of the data. This method allows you to restore only the data that has been affected. REORG and RUNSTATS utilities should be run against the tables and indexes containing this data.
- Return the application to its original state by re-BINDing it to the database.

The following are additional considerations when benchmarking on OS/2:

- If paging occurs during the scenario, ensure that SWAPPER.DAT has returned to the original size.
- Re-boot the system for repeatability, if necessary.

Output from the benchmark program should include an identifier for each test, the iteration of the program execution, the statement number, and the timing for the execution. A summary of benchmarking results after a series of measurements might look like the following:

```
Test     Iter.   Stmt    Timing         SQL Statement
Numbr    Numbr   Numbr   (hh:mm:ss.ss)
 002      05      01     00:00:01.34    CONNECT TO SAMPLE
 002      05      10     00:02:08.15    OPEN cursor_01
 002      05      15     00:00:00.24    FETCH cursor_01
 002      05      15     00:00:00.23    FETCH cursor_01
 002      05      15     00:00:00.28    FETCH cursor_01
 002      05      15     00:00:00.21    FETCH cursor_01
 002      05      15     00:00:00.20    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      20     00:00:00.84    CLOSE cursor_01
 002      05      99     00:00:00.03    CONNECT RESET
```

Figure 61. Benchmark Sample Results

Note: The data in the above report is shown for illustration purposes only. It does **not** represent measured results.

Examining this report would indicate that the CONNECT (statement 01) took 1.34 seconds, the OPEN CURSOR (statement 10) took 2 minutes and 8.15 seconds, the FETCHES (statement 15) returned seven rows with the longest delay being .28 seconds, the CLOSE CURSOR (statement 20) took .84 seconds, and the CONNECT RESET (statement 99) took .03 seconds.

It might be beneficial for your program to output your data in a delimited ASCII format so that it could later be imported into a database table or a spreadsheet for further statistical analysis.

Sample output for a benchmark report might be:

```
      PARAMETER        VALUES FOR EACH BENCHMARK TEST
      TEST NUMBER      001     002     003    004     005
      locklist         63      63      63      63      63
 >>   buffpage         1000    1175    1250    1325    1400     <<
      maxappls         8       8       8       8       8
      applheapsz       48      48      48      48      48
      dbheap           128     128     128     128     128
      sortheap         256     256     256     256     256
      maxlocks         22      22      22      22      22
      stmtheap         1024    1024    1024    1024    1024
      SQL STMT         AVERAGE TIMINGS (seconds)
        01             01.34   01.34   01.35   01.35   01.36
        10             02.15   02.00   01.55   01.24   01.00
        15             00.22   00.22   00.22   00.22   00.22
        20             00.84   00.84   00.84   00.84   00.84
        99             00.03   00.03   00.03   00.03   00.03
```

Figure 62. Benchmark Sample Timings Report

Note: The data in the above report is shown for illustration purposes only. It does **not** represent any measured results.

Examining the data in this example shows that changing the *buffpage* parameter successively lowered the OPEN CURSOR times from 2.15 seconds to 1.00 second. (The assumption is that there is only one (1) buffer pool with the size (NPAGES) set to -1. This means the size of the buffer pool is controlled by the *buffpage* parameter.)

In summary, the following steps/iterations may be followed to benchmark a database application:

Step 1 Leave the database and database manager tuning parameters at their **default** values except for:

- Those parameters significant to the workload and the objectives of the test. (You rarely have enough time to perform benchmark testing to tune all of the parameters, so you may want to start by using your best guess for some of the parameters and tune from that point.)
- Log sizes, which should be determined during unit and system testing of your application. (See “Size of Log Files (logfilsiz)” on page 519 for more information.)
- Any parameters that must be changed to enable your application to run (that is, the changes needed to prevent negative SQL return codes from such events as running out of memory for the statement heap).

Run your set of iterations for this initial case and calculate the average timing.

Step 2 Select one and only one tuning parameter to be tested, and change its value.

Step 3 Run another set of iterations and calculate the average timing.

Step 4 Depending on the results of the benchmark test, do one of the following:

- If performance improves, change the value of the same parameter and return to Step 3. Keep changing this parameter until the maximum benefit is shown.
- If performance degrades or remains unchanged, return the parameter to its previous value, return to Step 2, and select a new parameter. Repeat this procedure until all parameters have been tested.

  Note: If you were to graph the performance results, you would be looking for the point where the curve begins to plateau or decline.

You can write a driver program to help you with your benchmark testing. This driver program could be written using a language such as REXX or, for UNIX-based platforms, using shell scripts.

This driver program would execute the benchmark program, pass it the appropriate parameters, drive the test through multiple iterations, restore the environment to a consistent state, set up the next test with new parameter values, and collect/consolidate the test results. These driver programs can be flexible enough that they could be used to run the entire set of benchmark tests, analyze the results, and provide a report of the final and best parameter values for the given test.

Chapter 19. Configuring DB2

DB2 has been designed with an extensive array of tuning and configuration parameters. These parameters fall into two general categories:

- “Database Manager Parameters” on page 460
- “Database Parameters” on page 464.

In addition to descriptions of the individual parameters, the following topics are available:

- “Tuning Configuration Parameters”
- “Parameter Details by Function” on page 469 (each functional area has its own list of configuration parameters)
- “Establish Environment Variables and the Profile Registry” on page 60

  There may be performance-related environment variables for your specific platform that you should consider using in addition to the performance-related configuration parameters.
- Chapter 14, “Operational Performance” on page 395
- Chapter 18, “Benchmark Testing” on page 447.

You should review all of the parameter summaries in Table 36 on page 461 and Table 38 on page 466, and then focus on the descriptions and tuning of those which will provide you with the greatest benefit in your working environment.

Tuning Configuration Parameters

The disk space and memory allocated by the database manager on the basis of default values of the parameters may be sufficient to meet your needs in some situations, however, you may not be able to achieve maximum performance using these default values.

Since the default values are oriented towards machines with relatively small memory and dedicated as database servers, you may need to modify them if your environment has:

- Large databases
- Large numbers of connections
- High performance requirements for a specific application
- Unique query or transaction loads or types
- Different machine configuration or usage.

Each transaction processing environment is unique in one or more aspects. These differences can have a profound impact on the performance of the database manager when using the default configuration. For this reason, you are strongly advised to tune your configuration for your environment.

Different types of applications and users have different response time requirements and expectations. Applications could range from simple data entry screens to strategic

applications involving dozens of complex SQL statements accessing dozens of tables per unit of work. For example, response time requirements could vary considerably in a telephone customer service application versus a batch report generation application.

The other related topics can be used to help you benchmark your application to tune the configuration parameters:

- Database Manager Parameters
- Database Parameters
- "Parameter Details by Function" on page 469 (each functional area has its own list of configuration parameters)
- Chapter 14, "Operational Performance" on page 395
- Chapter 18, "Benchmark Testing" on page 447
- Database system monitor element descriptions in the *System Monitor Guide and Reference*.

Database Manager Parameters

Database manager parameters are stored in a file named `db2systm`. This file is created when the instance of the database manager is created. In UNIX-based environments, this file can be found in the `sqllib` subdirectory for the instance of the database manager. In all other environments, the default location of this file is the instance subdirectory of the `sqllib` directory. If the DB2INSTPROF variable is set, the file is in the `instance` subdirectory of the directory specified by the DB2INSTPROF variable.

In a parallel environment, this file resides on a shared file system so that all nodes have access to the same file. The configuration of database managers is the same on all nodes.

Most of the parameters either affect the amount of system resources that will be allocated to a single instance of the database manager, or they configure the setup of the database manager and the different communications subsystems based on environmental considerations. In addition, there are other parameters that serve informative purposes only and cannot be changed. All of these parameters have global applicability independent of any single database stored under that instance of the database manager.

The `db2systm` file cannot be directly edited. It can only be changed or viewed using a supplied API or by a tool which calls that API.

Attention: If you edit the file using a method other than those provided by the product, you may make your system unusable. We strongly recommend that you do not change this file using methods other than those documented and supported by DB2.

You may use one of the following three methods to reset, update, and view the database manager configuration parameters:

- Using the DB2 Control Center. The DB2 Control Center provides both the Configure Instance notebook and the Performance Configuration SmartGuide to alter the value of configuration parameters. This SmartGuide generates values to

parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database. See the on-line help available with the Control Center for information on using these interfaces.

- Using the command line processor. Commands to change the settings can be quickly and conveniently entered. See the *Command Reference* for more information about the following commands:
  - GET DATABASE MANAGER CONFIGURATION (or GET DBM CFG)
  - UPDATE DATABASE MANAGER CONFIGURATION (or UPDATE DBM CFG)
  - RESET DATABASE MANAGER CONFIGURATION (or RESET DBM CFG)
- Using the application programming interfaces (APIs). The APIs can easily be called from a host-language program. See the *API Reference* for more information.

After changing the parameters, the database manager must be stopped (`db2stop`) and then restarted (`db2start`) in order for the new parameter values to take effect. For clients, changes in the database manager configuration parameters take effect the next time the client connects to a server. While new parameter values are not immediately effective, viewing the parameter settings will always show the latest updates.

Database Manager Configuration Parameter Summary

The following table lists the parameters in the database manager configuration file for database servers. When changing the database manager configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

The column “Performance Impact” in the following table provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters; which in some cases, will mean that you accept the default provided.
- **Medium** — indicates the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focussed on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. While you do not have to tune these parameters for performance, they can be very important for other aspects of your system configuration, such as enabling communication support.

Table 36 (Page 1 of 4). Configurable Database Manager Configuration Parameters

Parameter	Performance Impact	Additional Information
agentpri	High	“Priority of Agents (agentpri)” on page 512

Table 36 (Page 2 of 4). Configurable Database Manager Configuration Parameters		
Parameter	**Performance Impact**	**Additional Information**
agent_stack_sz	Low	"Agent Stack Size (agent_stack_sz)" on page 488
aslheapsz	High	"Application Support Layer Heap Size (aslheapsz)" on page 492
authentication	Low	"Authentication Type (authentication)" on page 572
backbufsz	Medium	"Default Backup Buffer Size (backbufsz)" on page 475
comm_bandwidth	Medium	"Communications Bandwidth (comm_bandwidth)" on page 565
conn_elapse	Medium	"Connection Elapse Time (conn_elapse)" on page 556
cpuspeed	Low (see note)	"CPU Speed (cpuspeed)" on page 565
dft_account_str	None	"Default Charge-Back Account (dft_account_str)" on page 568
dft_client_adpt	None	"Default Client Adapter Number (dft_client_adpt)" on page 553
dft_client_comm	None	"Default Client Communication Protocol (dft_client_comm)" on page 552
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_uow*	Medium	"Default Database System Monitor Switches (dft_monswitches)" on page 564
dftdbpath	None	"Default Database Path (dftdbpath)" on page 573
diaglevel	Low	"Diagnostic Error Capture Level (diaglevel)" on page 562
diagpath	None	"Diagnostic Data Directory Path (diagpath)" on page 563
dir_cache	Medium	"Directory Cache Support (dir_cache)" on page 497
dir_obj_name	None	"Object Name in DCE Namespace (dir_obj_name)" on page 551
dir_path_name	None	"Directory Path Name in DCE Namespace (dir_path_name)" on page 550
dir_type	None	"Directory Services Type (dir_type)" on page 550
discover	Medium	"Discovery Mode (discover)" on page 554
discover_comm	Low	"Search Discovery Communications Protocols (discover_comm)" on page 555
discover_inst	Low	"Discover Server Instance (discover_inst)" on page 555
dos_rqrioblk	High	"DOS Requester I/O Block Size (dos_rqrioblk)" on page 494
drda_heap_sz	Low	"DRDA Heap Size (drda_heap_sz)" on page 486
fcm_num_anchors	High	"Number of FCM Message Anchors (fcm_num_anchors)" on page 556
fcm_num_buffers	High	"Number of FCM Buffers (fcm_num_buffers)" on page 557
fcm_num_connect	High	"Number of FCM Connection Entries (fcm_num_connect)" on page 558
fcm_num_rqb	High	"Number of FCM Request Blocks (fcm_num_rqb)" on page 558
fileserver	None	"IPX/SPX File Server Name (fileserver)" on page 548
indexrec	Medium	"Index Re-creation Time (indexrec)" on page 529
ipx_socket	None	"IPX/SPX Socket Number (ipx_socket)" on page 549

Table 36 (Page 3 of 4). Configurable Database Manager Configuration Parameters		
Parameter	**Performance Impact**	**Additional Information**
java_heap_sz	High	"Maximum Java Interpreter Heap Size (java_heap_sz)" on page 498
jdk11_path	None	"Java Development Kit 1.1 Installation Path (jdk11_path)" on page 569
keepdari	Medium	"Keep DARI Process Indicator (keepdari)" on page 517
maxagents	High	"Maximum Number of Agents (maxagents)" on page 514
maxcagents	High	"Maximum Number of Concurrent Agents (maxcagents)" on page 513
max_connretries	Medium	"Node Connection Retries (max_connretries)" on page 559
max_coordagents	High	"Maximum Number of Coordinating Agents (max_coordagents)" on page 515
maxdari	Medium	"Maximum Number of DARI Processes (maxdari)" on page 518
max_querydegree	High	"Maximum Query Degree of Parallelism (max_querydegree)" on page 559
max_time_diff	Medium	"Maximum Time Difference Among Nodes (max_time_diff)" on page 560
maxtotfilop	Medium	"Maximum Total Files Open per Application (maxtotfilop)" on page 511
min_priv_mem	Medium	"Minimum Committed Private Memory (min_priv_mem)" on page 489
mon_heap_sz	Low	"Database System Monitor Heap Size (mon_heap_sz)" on page 495
nname	None	"NetBIOS Workstation Name (nname)" on page 546
numdb	Low	"Maximum Number of Concurrently Active Databases (numdb)" on page 566
num_initagents	Medium	"Initial Number of Agents in Pool (num_initagents)" on page 516
num_poolagents	High	"Agent Pool Size (num_poolagents)" on page 515
objectname	None	"IPX/SPX DB2 Server Object Name (objectname)" on page 548
intra_parallel	High	"Enable Intra-Partition Parallelism (intra_parallel)" on page 560
priv_mem_thresh	Medium	"Private Memory Threshold (priv_mem_thresh)" on page 490
query_heap_sz	Medium	"Query Heap Size (query_heap_sz)" on page 485
restbufsz	Medium	"Default Restore Buffer Size (restbufsz)" on page 476
resync_interval	None	"Transaction Resync Interval (resync_interval)" on page 534
route_obj_name	None	"Routing Information Object Name (route_obj_name)" on page 552
rqrioblk	High	"Client I/O Block Size (rqrioblk)" on page 493
sheapthres	High	"Sort Heap Threshold (sheapthres)" on page 482
spm_log_file_sz	Low	"Sync Point Manager Log File Size (spm_log_file_sz)" on page 535
spm_max_resync	Low	"Sync Point Manager Resync Agent Limit (spm_max_resync)" on page 536

Table 36 (Page 4 of 4). Configurable Database Manager Configuration Parameters		
Parameter	**Performance Impact**	**Additional Information**
spm_name	None	"Sync Point Manager Name (spm_name)" on page 534
ss_logon	None	"LOGON Required for DB2START/DB2STOP (ss_logon)" on page 574
start_stop_time	Low	"Start and Stop Timeout (start_stop_time)" on page 561
svcename	None	"TCP/IP Service Name (svcename)" on page 547
sysadm_group	None	"System Administration Authority Group Name (sysadm_group)" on page 570
sysctrl_group	None	"System Control Authority Group Name (sysctrl_group)" on page 571
sysmaint_group	None	"System Maintenance Authority Group Name (sysmaint_group)" on page 572
tm_database	None	"Transaction Manager Database Name (tm_database)" on page 533
tp_mon_name	None	"Transaction Processor Monitor Name (tp_mon_name)" on page 567
tpname	None	"APPC Transaction Program Name (tpname)" on page 547
trust_allclnts	None	"Trust All Clients (trust_allclnts)" on page 574
trust_clntauth	None	"Trusted Clients Authentication (trust_clntauth)" on page 575
udf_mem_sz	Low	"UDF Shared Memory Set Size (udf_mem_sz)" on page 487
Note: The *cpuspeed* parameter can have a significant impact on performance but you should use the default value, except in very specific circumstances, as documented in the parameter description.		

Table 37. Informational Database Manager Configuration Parameters	
Parameter	**Additional Information**
nodetype	"Machine Node Type (nodetype)" on page 568
release	"Configuration File Release Level (release)" on page 536

Database Parameters

Parameters for an individual database are stored in a configuration file named `SQLDBCON`. This file is stored along with other control files for the database in the `SQLnnnnn` directory, where `nnnnn` is a number assigned when the database was created. (For more information about the location of this directory, see "Database Physical Directories" on page 25.) There is a single configuration file per database. Most of the parameters specify the amount of resources allocated to that database. In addition, there is descriptive information and flags indicating the status of the database.

In a partitioned database environment this file exists for each database on each database partition server, and specifies the database parameters for each node. If you want to have all the database partition servers (or a subset of them) share the same database configuration values, you should write a script or application to update the

multiple database configuration files. Each of the configuration files should then have the same values.

The SQLDBCON file cannot be directly edited, and can only be changed or viewed via a supplied API or by a tool which calls that API.

Attention: If you edit the file using a method other than those provided by DB2, you may make the database unusable. We strongly recommend that you do not change this file using methods other than those documented and supported by DB2.

You may use one of the following three methods to reset, update, and view the database configuration parameters:

- Using the Control Center. The DB2 Control Center provides both the Configure Database notebook and the Performance Configuration SmartGuide to alter the value of configuration parameters. This SmartGuide generates values to parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database. See the on-line help available with the Control Center for information on using these interfaces.
- Using the command line processor. Commands to change the settings can be quickly and conveniently entered. See the *Command Reference* for more information about the following commands:
  - GET DATABASE CONFIGURATION (or GET DB CFG)
  - UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (or RESET DB CFG)
- Using the application programming interfaces (APIs). The APIs can easily be called from a host-language program. (See the *API Reference* for more information.)

Updates to any changeable parameters will not take effect while applications are connected to the database. All applications must first disconnect from the database. (If the database was activated, then it must be deactived and reactived.) Then, at the first new connect to the database, the changes will take effect. You should note that some parameter changes, such as *newlogpath*, *logfilsiz* and *logprimary*, may take a noticeable amount of time to take effect due to the overhead associated with allocating space. You may wish to make a test connection to the database so the change will be made at the time of the test connection and any overhead will not affect other users. If you are concerned about the overhead as discussed here, consider using the ACTIVATE DATABASE command as found in the *Command Reference*.

Changing some database configuration parameters can influence the access plan chosen by the SQL optimizer. These database parameters are discussed in “Configuration Parameters Affecting Query Optimization” on page 301. After changing any of the parameters discussed there, you should consider rebinding your applications to ensure the best access plan is being used for your SQL statements.

While new parameter values are not immediately effective, viewing the parameter settings will always show the latest updates.

Note: A number of database configuration parameters (including *logretain* and *userexit*) are described as having acceptable values of either "Yes" or "No," or "On" or "Off" in the help and other DB2 books. To clarify what may be confusing, "Yes" should be considered equivalent to "On" and "No" should be considered equivalent to "Off."

Database Configuration Parameter Summary

The following table lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

The column "Performance Impact" in the following table provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters; which in some cases, will mean that you accept the default provided.
- **Medium** — indicates the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focussed on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. While you do not have to tune these parameters for performance, they can be very important for other aspects of your system configuration, such as enabling communication support.

Table 38 (Page 1 of 3). Configurable Database Configuration Parameters

Parameter	Performance Impact	Additional Information
adsm_mgmtclass	None	"ADSTAR Distributed Storage Manager Management Class (adsm_mgmtclass)" on page 531
adsm_nodename	None	"ADSTAR Distributed Storage Manager Node Name (adsm_nodename)" on page 532
adsm_owner	None	"ADSTAR Distributed Storage Manager Owner Name (adsm_owner)" on page 532
adsm_password	None	"ADSTAR Distributed Storage Manager Password (adsm_password)" on page 532
app_ctl_heap_sz	Medium	"Application Control Heap Size (app_ctl_heap_sz)" on page 480
applheapsz	Medium	"Application Heap Size (applheapsz)" on page 484
autorestart	Low	"Auto Restart Enable (autorestart)" on page 529
avg_appls	High	"Average Number of Active Applications (avg_appls)" on page 509
buffpage	High	"Buffer Pool Size (buffpage)" on page 470
catalogcache_sz	Medium	"Catalog Cache Size (catalogcache_sz)" on page 473
chngpgs_thresh	High	"Changed Pages Threshold (chngpgs_thresh)" on page 502

Table 38 (Page 2 of 3). Configurable Database Configuration Parameters		
Parameter	**Performance Impact**	**Additional Information**
copyprotect	None	"Copy Protection Enable (copyprotect)" on page 538
dbheap	Medium	"Database Heap (dbheap)" on page 472
dft_degree	High	"Default Degree (dft_degree)" on page 542
dft_extent_sz	Medium	"Default Extent Size of Table Spaces (dft_extent_sz)" on page 506
dft_loadrec_ses	Medium	"Default Number of Load Recovery Sessions (dft_loadrec_ses)" on page 530
dft_prefetch_sz	Medium	"Default Prefetch Size (dft_prefetch_sz)" on page 505
dft_queryopt	Medium	"Default Query Optimization Class (dft_queryopt)" on page 543
dir_obj_name	None	"Object Name in DCE Namespace (dir_obj_name)" on page 551
dft_sqlmathwarn	None	"Continue upon Arithmetic Exceptions (dft_sqlmathwarn)" on page 541
discover_db	Medium	"Discover Database (discover_db)" on page 554
dlchktime	Medium	"Time Interval for Checking Deadlock (dlchktime)" on page 499
estore_seg_sz	Medium	"Extended Storage Memory Segment Size (estore_seg_sz)" on page 507
indexrec	Medium	"Index Re-creation Time (indexrec)" on page 529
indexsort	Low (see note)	"Index Sort Flag (indexsort)" on page 504
locklist	High	"Maximum Storage for Lock List (locklist)" on page 477
locktimeout	Medium	"Lock Timeout (locktimeout)" on page 501
logbufsz	High	"Log Buffer Size (logbufsz)" on page 474
logfilsiz	Medium	"Size of Log Files (logfilsiz)" on page 519
logprimary	Medium	"Number of Primary Log Files (logprimary)" on page 521
logretain	Low	"Log Retain Enable (logretain)" on page 527
logsecond	Medium	"Number of Secondary Log Files (logsecond)" on page 522
maxappls	High	"Maximum Number of Active Applications (maxappls)" on page 508
maxfilop	Medium	"Maximum Database Files Open per Application (maxfilop)" on page 510
maxlocks	High	"Maximum Percent of Lock List Before Escalation (maxlocks)" on page 500
mincommit	High	"Number of Commits to Group (mincommit)" on page 525
newlogpath	Low	"Change the Database Log Path (newlogpath)" on page 523
num_estore_segs	Medium	"Number of Extended Storage Memory Segments (num_estore_segs)" on page 507
num_freqvalues	Low	"Number of Frequent Values Retained (num_freqvalues)" on page 544
num_iocleaners	High	"Number of Asynchronous Page Cleaners (num_iocleaners)" on page 502
num_ioservers	High	"Number of I/O Servers (num_ioservers)" on page 504

Table 38 (Page 3 of 3). Configurable Database Configuration Parameters		
Parameter	**Performance Impact**	**Additional Information**
num_quantiles	Low	"Number of Quantiles for Columns (num_quantiles)" on page 544
pckcachesz	High	"Package Cache Size (pckcachesz)" on page 479
rec_his_retentn	None	"Recovery History Retention Period (rec_his_retentn)" on page 531
seqdetect	High	"Sequential Detection Flag (seqdetect)" on page 505
softmax	Medium	"Recovery Range and Soft Checkpoint Interval (softmax)" on page 526
sortheap	High	"Sort Heap Size (sortheap)" on page 482
stat_heap_sz	Low	"Statistics Heap Size (stat_heap_sz)" on page 485
stmtheap	Medium	"Statement Heap Size (stmtheap)" on page 484
userexit	Low	"User Exit Enable (userexit)" on page 528
util_heap_sz	Low	"Utility Heap Size (util_heap_sz)" on page 475
Note: Changing the *indexsort* parameter to a value other than the default can have a negative impact on the performance of creating indexes. You should always try to use the default for this parameter.		

Table 39. Informational Database Configuration Parameters	
Parameter	**Additional Information**
backup_pending	"Backup Pending Indicator (backup_pending)" on page 539
codepage	"Code Page for the Database (codepage)" on page 538
codeset	"Codeset for the Database (codeset)" on page 537
collate_info	"Collating Information (collate_info)" on page 538
country	"Country code for the Database (country)" on page 537
database_consistent	"Database is Consistent (database_consistent)" on page 539
database_level	"Database Release Level (database_level)" on page 537
log_retain_status	"Log Retain Status Indicator (log_retain_status)" on page 540
loghead	"Log Head Identification (loghead)" on page 524
logpath	"Location of Log Files (logpath)" on page 524
multipage_alloc	"MultiPage File Allocation Enabled (multipage_alloc)" on page 540
nextactive	"Next Active Log (nextactive)" on page 524
numsegs	"Default Number of SMS Containers (numsegs)" on page 506
release	"Configuration File Release Level (release)" on page 536
restore_pending	"Restore Pending (restore_pending)" on page 540
rollfwd_pending	"Roll Forward Pending Indicator (rollfwd_pending)" on page 540
territory	"Territory for the Database (territory)" on page 537
user_exit_status	"User Exit Status Indicator (user_exit_status)" on page 540

Parameter Details by Function

This following sections provide additional details to assist in understanding and tuning the different configuration parameters. This discussion of the individual parameters is organized based on their function or purpose:

- "Capacity Management"
- "Logging and Recovery" on page 519
- "Database Management" on page 536
- "Communications" on page 546
- "Parallel" on page 555
- "Instance Management" on page 562.

The discussion of each parameter includes the following information:

Configuration Type Indicates which configuration file contains the setting for the parameter:

- Database manager (which affects an instance of the database manager and all databases defined within that instance)
- Database (which affects a specific database)

Parameter Type Indicates whether or not you can change the parameter value:

- *Configurable*

  A range of values are possible and the parameter may need to be tuned based on the database administrator's knowledge of the applications and/or from benchmarking experience.

- *Informational*

  These parameters are changed only by the database manager itself and will contain information such as the release of DB2 that a database was created under or an indication that a required backup is pending.

Capacity Management

There are a number of configuration parameters at both the database and database manager levels that can impact the throughput on your system. These parameters are categorized in the following groups:

- "Database Shared Memory" on page 470
- "Application Shared Memory" on page 480
- "Agent Private Memory" on page 481
- "Agent/Application Communication Memory" on page 491
- "Database Manager Instance Memory" on page 495
- "Locks" on page 499
- "I/O and Storage" on page 502
- "Agents" on page 508

- “Database Application Remote Interface (DARI)” on page 517

For an introduction to DB2's memory management, see “How DB2 Uses Memory” on page 395.

Database Shared Memory

The following parameters affect the database global memory allocated on your system:

- “Buffer Pool Size (buffpage)”
- “Database Heap (dbheap)” on page 472.
- “Catalog Cache Size (catalogcache_sz)” on page 473.
- “Log Buffer Size (logbufsz)” on page 474.
- “Utility Heap Size (util_heap_sz)” on page 475.
- “Default Backup Buffer Size (backbufsz)” on page 475.
- “Default Restore Buffer Size (restbufsz)” on page 476.
- “Maximum Storage for Lock List (locklist)” on page 477.
- “Package Cache Size (pckcachesz)” on page 479.
- “Sort Heap Size (sortheap)” on page 482. This parameter only affects database global memory if you have shared sorts.

See “How DB2 Uses Memory” on page 395 for information about how database global memory relates to the rest of the memory allocated by the database manager.

Buffer Pool Size (buffpage)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX	1000 [2*maxappls - 524 288]
OS/2 and NT	250 [2*maxappls - 524 288]

Unit of Measure Pages (4KB)

When Allocated When the first application connects to the database

When Freed When last application disconnects from the database

Related Parameters

- “Changed Pages Threshold (chngpgs_thresh)” on page 502
- “Database Heap (dbheap)” on page 472
- “Number of Asynchronous Page Cleaners (num_iocleaners)” on page 502

Each database has at least one buffer pool (IBMDEFAULTBP, which is created when the database is created), and can have more. All buffer pools reside in global memory, which is available to all applications using the database. The memory is allocated on the machine where the database is located. If the buffer pools are large enough to keep the required data in memory, less disk activity will occur. Conversely, if the buffer pools are not large enough, the overall performance of the database can be severely curtailed

and the database manager can become I/O-bound as a result of a high amount of disk activity (I/O) required to process the data your application requires.

The *buffpage* parameter controls the size of a buffer pool when the CREATE BUFFERPOOL or ALTER BUFFERPOOL statement was run with NPAGES -1; otherwise, the *buffpage* parameter is ignored and the buffer pool will be created with the number of pages specified by the NPAGES parameter.

To determine whether the *buffpage* parameter is active for a buffer pool, do a `SELECT * from SYSIBM.SYSBUFFERPOOLS`. Each buffer pool that has an NPAGES value of -1 uses *buffpage*.

Notes:

1. When a database is created in DB2 Version 5, one buffer pool (IBMDEFAULTBP) is automatically created, and it's NPAGES is set to 1000 for UNIX-based platforms, and 250 for all other platforms.
2. When a database is migrated to DB2 Version 5, one buffer pool (IBMDEFAULTBP) is automatically created, and it's NPAGES is set to -1.

There is a trade-off between the buffer pool size and the memory allocations of other system users. Memory requirements of database servers are so important on multi-user high transaction rate servers, that database servers and file or communication servers are often separated and reside on different machines.

All buffer pools are allocated when the first application connects to the database, or when the database is explicitly activated. As an application requests data out of the database, pages containing that data are transferred to one of the buffer pools from disk. (Note that database data is stored in pages within the tables on the disk.) Pages are not written back to disk until one of the following occurs:

- All applications disconnect from the database
- The database is explicitly deactivated
- The database quiesces (that is, all connected applications have committed)
- Another page needs to be read into the buffer pool
- A page cleaner is available (*num_iocleaners*) and is activated by the database manager.

Recommendations:

- Instead of using the *buffpage* configuration parameter, you can use the CREATE BUFFERPOOL and ALTER BUFFERPOOL SQL statements to change buffer pools and their sizes.
- The size of the buffer pool is used by the optimizer in determining access plans. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.
- Because this parameter can have a major impact on performance, you should consider the following factors to ensure that excessive page swapping does not occur:
  - The amount of installed memory on your machine.

  - The memory required by other applications running concurrently with the database manager on the same machine.

  Page swapping results when there is not enough memory to hold the page that is being accessed. The result is that the page is written ("swapped") to temporary disk storage to make room for the other page. When the page on the temporary disk storage is needed, it is "swapped back" into memory.
- You may wish to allocate as much as 75% of the machine's memory to the database buffer pools when you have the following:
  - Multiple users
  - A machine used only as a database server
  - A large amount of repeated access to the same data and index pages
  - One database on the machine.
- For every buffer pool page allocated, some space is used in the database heap for internal control structures.

  If the total size of the buffer pool (or buffer pools) is increased, you may also need to increase *dbheap*.

You may use the database system monitor to calculate the buffer pool hit ratio, which can help you tune your buffer pools. See the *System Monitor Guide and Reference*.

Database Heap (dbheap)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX 1200 [32 – 60 000]

OS/2 and NT Database Server with local and remote clients
600 [32 – 60 000]

OS/2 and NT Database Server with local clients
300 [32 – 60 000]

Unit of Measure Pages (4KB)

When Allocated First connection to the database

When Freed When last application disconnects from the database

Related Parameters

- "Catalog Cache Size (catalogcache_sz)" on page 473
- "Log Buffer Size (logbufsz)" on page 474

There is one database heap per database, and the database manager uses it on behalf of all applications connected to the database. It contains control block information for tables, indexes, table spaces, buffer pools, event monitor buffers, as well as space for the log buffer (*logbufsz*) and the catalog cache (*catalogcache_sz*). Therefore, the size of the heap will be dependent on the number of control blocks stored in the heap at a

given time. The control block information is kept in the heap until all applications disconnect from the database.

The minimum amount the database manager needs to get started is allocated at the first connection. The data area is expanded as needed up to the maximum specified by *dbheap*.

Recommendation: This value will need to be increased when an application receives an error indicating that there is not enough storage available in the database heap to process the statement.

You may use the database system monitor to track the highest amount of memory that was used for the database heap. See the *db_heap_top (maximum database heap allocated)* monitor element description in the *System Monitor Guide and Reference* for more information.

When setting this parameter, you should consider:

- The value of *logbufsz*, because the log buffer is allocated from the database heap.
- The value of *catalogcache_sz*, because the catalog cache is allocated from the database heap.

Catalog Cache Size (catalogcache_sz)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX 64 [1 – *dbheap*]

OS/2 and NT Database Server with local and remote clients
32 [1 – *dbheap*]

OS/2 and NT Database Server with local clients
16 [1 – *dbheap*]

Unit of Measure Pages (4KB)

Related Parameters

- “Database Heap (dbheap)” on page 472
- “Log Buffer Size (logbufsz)” on page 474

This parameter indicates the maximum amount of space that the catalog cache can use from the database heap (*dbheap*). The catalog cache is used to store table descriptor information that is used when a table, view or alias is referenced during the compilation of an SQL statement.

Use of this cache can help improve performance of binding SQL statements (including dynamic SQL), if the same tables, views, or aliases have been referenced in previous statements.

Running any DDL statements against a table will purge that table's entry in the catalog cache. Otherwise a table entry is kept in the cache until space is needed for a different table, but it will not be removed from the cache until any units of work referencing that table have completed.

Recommendation: Start with the default value and tune it by using the database system monitor.

See the *System Monitor Guide and Reference* for information about the following monitor elements:

- *cat_cache_lookups (catalog cache lookups)*
- *cat_cache_inserts (catalog cache inserts)*
- *cat_cache_overflows (catalog cache overflows)*
- *cat_cache_heap_full (catalog cache heap full)*

These database system monitor elements can help you determine whether you should adjust this configuration parameter. When tuning this parameter, you should increase it in small increments, for example, two pages at a time.

Note: The catalog cache only exists at the catalog node in a multinode environment.

In general, more cache space is required if a unit of work contains several dynamic SQL statements or if you are binding packages that contain a lot of static SQL statements.

When you set the size of the catalog cache, also consider the size of the log files (*logbufsz*), because both *catalogcache_sz* and *logbufsz* are allocated from the database heap (*dbheap*).

Log Buffer Size (logbufsz)

Configuration Type Database

Parameter Type Configurable

Default [Range] 8 [4 – 128]

Unit of Measure Pages (4KB)

Related Parameters

- “Catalog Cache Size (catalogcache_sz)” on page 473
- “Database Heap (dbheap)” on page 472
- “Number of Commits to Group (mincommit)” on page 525

This parameter allows you to specify the amount of the database heap (defined by the *dbheap* parameter) to use as a buffer for log records before writing these records to disk. The log records are written to disk when one of the following occurs:

- A transaction commits or a group of transactions commit, as defined by the *mincommit* configuration parameter
- The log buffer is full
- As a result of some other internal database manager event.

This parameter must also be less than or equal to the *dbheap* parameter. Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently and more log records will be written at each time.

Recommendation: Increase the size of this buffer area if there is considerable read activity on a dedicated log disk, or there is high disk utilization. When increasing the value of this parameter, you should also consider the *dbheap* parameter since the log buffer area uses space controlled by the *dbheap* parameter.

You may use the database system monitor to determine how much of the log buffer space is used for a particular transaction (or unit of work).

For more information see the *log_space_used* (unit of work log space used) monitor element description in the *System Monitor Guide and Reference*. .

When you set the log buffer size, also consider the size of the catalog cache (*catalogcache_sz*), because both *logbufsz_sz* and *catalogcache_sz* are allocated from the database heap (*dbheap*).

Utility Heap Size (util_heap_sz)

Configuration Type Database

Parameter Type Configurable

Default [Range] 5000 [16 – 524 288]

Unit of Measure Pages (4KB)

When Allocated As required by the database manager utilities

When Freed When the utility no longer needs the memory

Related Parameters

- “Default Backup Buffer Size (backbufsz)”
- “Default Restore Buffer Size (restbufsz)” on page 476

This parameter indicates the maximum amount of memory that can be used simultaneously by the BACKUP, RESTORE and LOAD and load recovery utilities.

Recommendation: Use the default value unless your utilities run out of space, in which case you should increase this value. If memory on your system is constrained, you may wish to lower the value of this parameter to limit the memory used by the database utilities. If the parameter is set too low, you may not be able to concurrently run utilities. You need to set this parameter large enough to accommodate all of the buffers that you want to allocate for the concurrent utilities.

Default Backup Buffer Size (backbufsz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 1024 [16 – 524 288]

Unit of Measure Pages (4KB)

When Allocated When the backup utility is called

When Freed When the backup utility completes its processing

Related Parameters

- "Default Restore Buffer Size (restbufsz)"
- "Utility Heap Size (util_heap_sz)" on page 475

This parameter specifies the size of the buffer used when backing up the database if the buffer size is not explicitly specified when calling the backup utility. For more information about the backup utility, see the *Command Reference*. .

When backing up a database, the data is first copied to an internal buffer. Data is then written from this buffer to the backup media when the buffer is full.

Tuning this buffer size can help improve the performance of the backup utility as well as minimize the impact on the performance of other concurrent database operations.

Default Restore Buffer Size (restbufsz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 1024 [16 – 524 288]

Unit of Measure Pages (4KB)

When Allocated When the restore utility is called

When Freed When the restore utility completes its processing

Related Parameters

- "Default Backup Buffer Size (backbufsz)" on page 475
- "Utility Heap Size (util_heap_sz)" on page 475

This parameter specifies the size of the buffer used when restoring the database if a buffer size is not explicitly specified when calling the restore utility. For more information about the restore utility, see the *Command Reference*.

When restoring a database, the data is first copied from the backup media to an internal buffer. Data is then written from this buffer to the target database media when the buffer is full.

Tuning this buffer size can help improve the performance of the restore database utility as well as minimize the impact on the performance of other concurrent database operations.

Maximum Storage for Lock List (locklist)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX 100 [4 – 60 000]

OS/2 and NT Database Server with local and remote clients
50 [4 – 60 000]

OS/2 and NT Database Server with local clients
25 [4 – 60 000]

Unit of Measure Pages (4KB)

When Allocated When the first application connects to the database

When Freed When last application disconnects from the database

Related Parameters

- "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 500
- "Maximum Number of Active Applications (maxappls)" on page 508

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database and it contains the locks held by all applications concurrently connected to the database. Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked.

Each lock requires 32 or 64 bytes of the lock list, depending on whether other locks are held on the object:

- 64 bytes are required to hold a lock on an object that has no other locks held on it
- 32 bytes are required to record a lock on an object that has an existing lock held on it.

When the percentage of the lock list used by one application reaches *maxlocks*, the database manager will perform lock escalation, from row to table, for the locks held by the application (described below). Although the escalation process itself does not take much time, locking entire tables (versus individual rows) decreases concurrency, and

overall database performance may decrease for subsequent accesses against the affected tables. Suggestions of how to control the size of the lock list are:

- Perform frequent COMMITs to release locks.
- When performing many updates, lock the entire table before updating (using the SQL LOCK TABLE statement). This will use only one lock, keeps others from interfering with the updates, but does reduce concurrency of the data. Use of the Repeatable Read isolation level may result in an automatic table lock. For more information on isolation levels, see Chapter 9, “Application Considerations” on page 265.
- Use the Cursor Stability isolation level when possible to decrease the number of share locks held. If application integrity requirements are not compromised use Uncommitted Read instead of Cursor Stability to further decrease the amount of locking.

Once the lock list is full, performance can degrade since lock escalation will generate more table locks and fewer row locks, thus reducing concurrency on shared objects in the database. Additionally there may be more deadlocks between applications (since they are all waiting on a limited number of table locks), which will result in transactions being rolled back. Your application will receive an SQLCODE of -912 when the maximum number of lock requests has been reached for the database.

Recommendation: If lock escalations are causing performance concerns you may need to increase the value of this parameter or the *maxlocks* parameter. You may use the database system monitor to determine if lock escalations are occurring.

For more information see the *lock_escals (lock escalations)* monitor element description in the *System Monitor Guide and Reference*.

The following steps may help in determining the number of pages required for your lock list:

1. Calculate a lower bound for the size of your lock list:

   ```
   (512 * 32 * maxappls) / 4096
   ```

   where 512 is an estimate of the average number of locks per application and 32 is the number of bytes required for each lock against an object that has an existing lock.

2. Calculate an upper bound for the size of your lock list:

   ```
   (512 * 64 * maxappls) / 4096
   ```

   where 64 is the number of bytes required for the first lock against an object.

3. Estimate the amount of contention you will have against your data and based on your expectations, choose an initial value for *locklist* that falls between the upper and lower bounds that you have calculated.

4. Using the database system monitor, as described below, tune the value of this parameter.

You may use the database system monitor to determine the maximum number of locks held by a given transaction.

For more information see the *locks_held_top (maximum number of locks held)* monitor element description in the *System Monitor Guide and Reference*. .

This information can help you validate or adjust the estimated number of locks per application. In order to perform this validation, you will have to sample several applications, noting that the monitor information is provided at a transaction level, not an application level.

You may also want to increase *locklist* if *maxappls* is increased, or if the applications being run perform infrequent commits.

You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

For more information on application performance and influencing query optimization, see Part 3, “Tuning Application Performance” on page 263.

Package Cache Size (pckcachesz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	-1 [-1, 32 – 64000]
Unit of Measure	Pages (4KB)
When Allocated	When the database is initialized
When Freed	When the database is shutdown

This parameter is allocated out of the database global memory, and is used for caching static and dynamic SQL statements on a database. There is one package cache for each database partition.

Caching packages allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs when reloading a package; or, in the case of dynamic SQL, eliminating the need for compilation. Sections are kept in the package cache until one of the following occurs:

- The database is shut down
- The package or dynamic SQL statement is invalidated
- The cache runs out of space.

This caching of the section for a static or dynamic SQL statement can improve performance especially when the same statement is used multiple times by applications connected to a database. This is particularly important in a transaction processing application.

By taking the default (-1) in a server or partitioned database environment, the value used to calculate the page allocation is eight times the value specified for the *maxappls* configuration parameter. The exception to this occurs if eight times *maxappls* is less than 32. In this situation, the default value of -1 will set *pckcachesz* to 32.

Recommendation: When tuning this parameter, you should consider whether the extra memory being reserved for the package cache might be more effective if it was allocated for another purpose, such as the bufferpool. For this reason, you should use benchmarking techniques when tuning this parameter.

Tuning this parameter is particularly important when several sections are used initially and then only a few are run repeatedly. If the cache is too large, memory is wasted holding copies of the initial sections.

See the *System Monitor Guide and Reference* for information about the following monitor elements:

- *pkg_cache_lookups (package cache lookups)*
- *pkg_cache_inserts (package cache inserts)*

These database system monitor elements can help you determine whether you should adjust this configuration parameter.

Note: The package cache is a working cache, so you cannot set this parameter to zero. There must be sufficient memory allocated in this cache to hold all sections of the SQL statements currently being executed. If there is more space allocated than currently needed, then sections are cached. These sections can simply be executed the next time they are needed without having to load or compile them. In this way the memory allocated by this parameter is part heap and part cache.

Application Shared Memory

The following parameter specifies the work area that is used by all agents (both coordinating and subagents) that work for an application:

- "Application Control Heap Size (app_ctl_heap_sz)"

Application Control Heap Size (app_ctl_heap_sz)

Configuration Type Database

Parameter Type Configurable

Default [Range]

Database Server with local and remote clients
128 [1–64 000]

Database Server with local clients
64 [1–64 000]

Partitioned Database Server with local and remote clients
256 [1–64 000]

Unit of Measure	Pages (4KB)
When Allocated	When an application starts
When Freed	When an application completes

This parameter determines the maximum size, in 4 KB pages, for the application control shared memory. Application control heaps are allocated from this shared memory.

One application control heap is allocated for each application per database partition for each database partition on which the application is active. The heap is allocated during connect processing by the first agent to receive a request for the application at each database partition. The heap is required to share information between agents working on behalf of the same application at a database partition.

Notes:

1. In a partitioned database environment, this heap is used to store copies of the executing sections of SQL statements for agents and subagents. Symmetric multiprocessor agents (SMP) subagents, however, use *applheapsz*, as do agents in all other environments.
2. Allocation will take place for an SMP machine that has the *parallel_enable* parameter set on.

Recommendation: Initially, start with the default value. You may have to set the value higher if you are running complex applications, or if you have a system that contains a large number of database partitions.

Agent Private Memory

The following parameters affect the amount of memory used for each database agent:

- “Sort Heap Size (sortheap)” on page 482.
- “Sort Heap Threshold (sheapthres)” on page 482.
- “Statement Heap Size (stmtheap)” on page 484.
- “Application Heap Size (applheapsz)” on page 484.
- “Statistics Heap Size (stat_heap_sz)” on page 485.
- “Query Heap Size (query_heap_sz)” on page 485.
- “DRDA Heap Size (drda_heap_sz)” on page 486.
- “UDF Shared Memory Set Size (udf_mem_sz)” on page 487.
- “Agent Stack Size (agent_stack_sz)” on page 488.
- “Minimum Committed Private Memory (min_priv_mem)” on page 489.
- “Private Memory Threshold (priv_mem_thresh)” on page 490.
- “Maximum Java Interpreter Heap Size (java_heap_sz)” on page 498. On UNIX-based platforms, *java_heap_sz* is allocated per agent.

See “How DB2 Uses Memory” on page 395 for information about how the private agent memory relates to the rest of the memory allocated by the database manager.

Sort Heap Size (sortheap)

Configuration Type Database

Parameter Type Configurable

Default [Range] 256 [16 – 524 288]

Unit of Measure Pages (4KB)

When Allocated As needed to perform sorts

When Freed When sorting is complete

Related Parameters "Sort Heap Threshold (sheapthres)"

This parameter defines the maximum number of private memory pages to be used for private sorts, or the maximum number of shared memory pages to be used for shared sorts. If the sort is a private sort, then this parameter affects agent private memory. If the sort is a shared sort, then this parameter affects the database shared memory. Each sort has a separate sort heap that is allocated as needed, by the database manager. This sort heap is the area where data is sorted. If directed by the optimizer, a smaller sort heap than the one specified by this parameter is allocated using information provided by the optimizer.

Recommendation:

- Appropriate indexes can minimize the use of the sort heap.
- Increase the size of this parameter when frequent large sorts are required.
- When increasing the value of this parameter, you should examine whether the *sheapthres* parameter in the database manager configuration file also needs to be adjusted.
- The sort heap size is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Sort Heap Threshold (sheapthres)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX 20 000 [250 – 2 097 152]

OS/2 and NT 10 000 [250 – 2 097 152]

Unit of Measure Pages (4KB)

Related Parameters "Sort Heap Size (sortheap)"

This parameter is used to control the total amount of memory that can be allocated across the database manager instance for sort heaps. When the total amount of memory allocated to all sort heaps exceeds the threshold, the maximum sort heap that can be allocated to subsequent requests will be reduced.

For private sorts, this threshold is not a hard limit on the amount of memory that can be allocated to sort heaps. Additional sort heap requests are not prevented from allocating more memory than the threshold. Instead, it is a trigger point beyond which sort heap allocation amounts will be constrained.

For shared sorts, however, this threshold is a hard limit. Memory for shared sorts is preallocated when the database is started, based on the sort heap threshold. Memory allocation for shared sorts is limited by the sort heap (*sortheap*) database configuration parameter. As the shared memory is consumed, the sort heap allocation is reduced to satisfy the sort memory request. The total memory for shared sorts for a database, however, will not exceed the sort heap threshold. An error will occur when the memory for shared sorts is exhausted.

Explicit definition of the threshold prevents the database manager from using excessive amounts of memory for large numbers of sorts.

Recommendation: Ideally, you should set this parameter to a reasonable multiple of the largest *sortheap* parameter you have in your database manager instance. This parameter should be **at least** two times the largest *sortheap* defined for any database within the instance.

If you are doing private sorts and your system is not memory constrained, an ideal value for this parameter can be calculated using the following steps:

1. Calculate the typical sort heap usage for each database:

   ```
     (typical number of concurrent agents running against the database)
   * (sortheap, as defined for that database)
   ```

2. Calculate the sum of the above results, which provides the total sort heap that could be used under typical circumstances for all databases within the instance.

For information about performing sorts in an SMP environment, see “Parallel Sort Strategies” on page 374.

You should use benchmarking techniques to tune this parameter to find the proper balance between sort performance and memory usage. See Chapter 18, “Benchmark Testing” on page 447 for more information. Also see “Sorting” on page 412 for more information on sorting.

You can use the database system monitor to track the sort activity.

For more information see the following monitor element description in the *System Monitor Guide and Reference*:

- *post_threshold_sorts (post threshold sorts)*

Statement Heap Size (stmtheap)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	2048 [128 – 60 000]
Unit of Measure	Pages (4KB)
When Allocated	For each statement during precompiling or binding
When Freed	When precompiling or binding of each statement is complete

The statement heap is used as a work space for the SQL compiler during compilation of an SQL statement. This parameter specifies the size of this work space.

This area does not stay permanently allocated, but is allocated and released for every SQL statement handled. Note that for dynamic SQL statements, this work area will be used during execution of your program; whereas, for static SQL statements, it is used during the bind process but not during program execution.

Recommendation: In most cases the default value of this parameter will be acceptable. If you have very large SQL statements and the database manager issues an error (that the statement is too complex) when it attempts to optimize a statement, you should increase the value of this parameter in regular increments (such as 256 or 1024) until the error situation is resolved.

Application Heap Size (applheapsz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	128 [16 – 60 000] 64 [16 – 60 000] (multinode)
Unit of Measure	Pages (4KB)
When Allocated	When an agent is initialized to do work for an application
When Freed	When an agent completes the work to be done for an application
Related Parameters	“Application Control Heap Size (app_ctl_heap_sz)” on page 480

This parameter defines the number of private memory pages available to be used by the database manager on behalf of a specific agent or subagent.

The heap is allocated when an agent or subagent is initialized for an application. The amount allocated will be the minimum amount needed to process the request given to the agent or subagent. As the agent or subagent requires more heap space to process larger SQL statements, the database manager will allocate memory as needed, up to the maximum specified by this parameter.

Note: In a partitioned database environment, the application control heap (*app_ctl_heap_sz*) is used to store copies of the executing sections of SQL statements for agents and subagents. SMP subagents, however, use *applheapsz*, as do agents in all other environments.

Recommendation: Increase the value of this parameter if your applications receive an error indicating that there is not enough storage in the application heap.

The application heap (*applheapsz*) is allocated out of agent private memory.

Statistics Heap Size (stat_heap_sz)

Configuration Type Database

Parameter Type Configurable

Default [Range] 4384 [1096 – 524 288]

Unit of Measure Pages (4KB)

When Allocated When the RUNSTATS utility is started

When Freed When the RUNSTATS utility is completed

Related Parameters

- "Number of Frequent Values Retained (num_freqvalues)" on page 544
- "Number of Quantiles for Columns (num_quantiles)" on page 544

This parameter indicates the **maximum** size of the heap used in collecting statistics using the RUNSTATS command.

Recommendation: The default value is appropriate when no distribution statistics are collected or when distribution statistics are only being collected for relatively narrow tables. The minimum value is **not** recommended when distribution statistics are being gathered, as only tables containing 1 or 2 columns will fit in the heap.

You should adjust this parameter based on the number of columns for which statistics are being collected. Narrow tables, with relatively few columns, require less memory for distribution statistics to be gathered. Wide tables, with many columns, require significantly more memory. If you are gathering distribution statistics for tables which are very wide and require a large statistics heap, you may wish to collect the statistics during a period of low system activity so you do not interfere with the memory requirements of other users.

Query Heap Size (query_heap_sz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients

Parameter Type	Configurable
Default [Range]	1000 [2 – 524 288]
Unit of Measure	Pages (4KB)
When Allocated	When an application connects to the database
When Freed	When the application disconnects from the database, or detaches from the instance
Related Parameters	“Application Support Layer Heap Size (aslheapsz)” on page 492

This parameter specifies the **maximum** amount of memory that can be allocated for the query heap. A query heap is used to store each query in the agent's private memory. The information for each query consists of the input and output SQLDA, the statement text, the SQLCA, the package name, creator, section number, and consistency token. This parameter is provided to ensure that an application does not consume unnecessarily large amounts of virtual memory within an agent.

The query heap is also used as the source of memory for the memory allocated for blocking cursors. This memory consists of a cursor control block and a fully resolved output SQLDA.

The initial query heap allocated will be the same size as the application support layer heap, as specified by the *aslheapsz* parameter. The query heap size must be greater than or equal to two (2), and must be greater than or equal to the *aslheapsz* parameter. If this query heap is not large enough to handle a given request, it will be reallocated to the size required by the request (not exceeding *query_heap_sz*). If this new query heap is more than 1.5 times larger than *aslheapsz*, the query heap will be reallocated to the size of *aslheapsz* when the query ends.

Recommendation: In most cases the default value will be sufficient. As a minimum, you should set *query_heap_sz* to a value at least five times larger than *aslheapsz*. This will allow for queries larger than *aslheapsz* and provide additional memory for three or four blocking cursors to be open at a given time.

If you have very-large LOBs, you may need to increase the value of this parameter so the query heap will be large enough to accommodate those LOBs.

DRDA Heap Size (drda_heap_sz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 128 [16 – 60 000]

Unit of Measure Pages (4KB)

When Allocated

- The DRDA Application Server (AS) allocates a DRDA heap each time a DRDA Application Requester(AR) connects to a DB2 database
- DB2 Connect allocates a DRDA heap each time it connects to a DRDA AS.

When Freed When a DRDA AR disconnects from the database

This parameter indicates the number of pages to allocate for the memory used by DB2 Connect and the DRDA Application Server Support Feature. The following items affect the amount of memory allocated out of this heap:

- The number of cursors opened by an application
- The number of input host variables
- The number of items in the select list
- The size of input and output data
- The length of SQL statements being bound or prepared.

Recommendation: Use the default value unless you receive an error code indicating that you do not have enough DRDA heap.

UDF Shared Memory Set Size (udf_mem_sz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 256 [128 – 60 000]

Unit of Measure Pages (4KB)

When Allocated When a UDF starts

When Freed When a UDF completes

This parameter is common to both fenced and unfenced User Defined Functions (UDFs). For a fenced UDF, it specifies the default allocation for memory to be shared between the database process and the UDF. In a non-partitioned database environment, there is only one shared memory set. In a partitioned database environment, there is a shared memory set for each database partition server, and all application agents and sub-agents running on that server use the same shared memory set.

For an unfenced UDF it specifies the size of the private memory set. In a non-partitioned database environment, the heap is allocated from private memory. In a partitioned database environment, the heap is allocated from the Application Controlled Shared memory for each node, and all application agents and subagents running on that node use the same shared memory set.

For both fenced and unfenced UDFs, this memory is used to pass data to a UDF and back to a database.

If no UDFs are used in applications, the memory is not allocated. If both fenced and unfenced UDFs are running in the same application, two memory allocations result: one for fenced UDFs, and one for unfenced UDFs.

For more information about user-defined functions, see the *Embedded SQL Programming Guide* and the *SQL Reference*.

Recommendation: The default setting should be adequate for all cases not involving the passing of LOB data to a UDF. For cases which pass LOB data to a UDF, you may need to increase the amount of memory allocated. You should set the value of this parameter at least 2 pages larger than the size of the input arguments and the result of the external function.

Note: The memory requirement for UDFs tends to be additive, so the number of UDFs referenced in an application affects the size of the default to be set for this configuration parameter.

Agent Stack Size (agent_stack_sz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range]

OS/2	64 [8 – 1000]
NT	16 [8 – 1000]

Unit of Measure Pages (4KB)

When Allocated When an agent is initialized to do work for an application

When Freed When an agent completes the work to be done for an application

The agent stack is the virtual memory that is allocated by DB2 for each agent. This memory is committed when it is required to process an SQL statement. You can use this parameter to optimize memory utilization of the server for a given set of

applications. More complex queries will use more stack space, compared to the space used for simple queries.

This parameter does not apply to UNIX-based platforms.

Recommendation: In most cases you should be able to use the default stack size. Only if your environment includes many highly complex queries should you need to increase the value of this parameter. If the stack size is not large enough to process your SQL statement, an error entry will be logged to the db2diag.log file, and an SQL code will be issued. You need to increase *agent_stack_sz* and restart the database instance.

You may be able to reduce the stack size in order to make more address space available to other clients, if your environment matches the following:

- Contains only simple applications (for example light OLTP), in which there are never complex queries
- Requires a relatively large number of concurrent clients (for example, more that 100).

The agent stack size and the number of concurrent clients are inversely related. For example, a larger stack size reduces the potential number of concurrent clients that can be running.

Minimum Committed Private Memory (min_priv_mem)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 32 [32 – 112 000]

Unit of Measure Pages (4KB)

When Allocated When the database manager is started

When Freed When the database manager is stopped

Related Parameters “Private Memory Threshold (priv_mem_thresh)” on page 490

This parameter specifies the number of pages that the database server process will reserve as private virtual memory, when a database manager instance is started (`db2start`). If the server requires more private memory, it will try to obtain more from the operating system when required.

This parameter does not apply to UNIX-based systems.

Recommendation: Use the default value.

You should only change the value of this parameter if you want to commit more memory to the database server. This action will save on allocation time. You should be careful, however, that you do not set that value too high, as it can impact the performance of non-DB2 applications.

Private Memory Threshold (priv_mem_thresh)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 1296 [-1; 32 – 112 000]

Unit of Measurement Pages (4KB)

Related Parameters “Minimum Committed Private Memory (min_priv_mem)” on page 489

This parameter is used to determine the amount of unused agent private memory that will be kept allocated, ready to be used by new agents that are started. It does not apply to UNIX-based platforms.

When an agent is terminated, instead of automatically deallocating all of the memory that was used by that agent, the database manager will only deallocate *excess memory allocations*, which is determined by the following formula:

```
Private memory allocated -
(private memory used + priv_mem_thresh)
```

If this formula produces a negative result, no action will be taken.

The following table provides an example to illustrate when memory will be allocated and deallocated. This example uses 100, as a arbitrary setting for *priv_mem_thresh*.

Description of Action	Memory Allocated	Memory Used
A number of agents are running and have allocated memory.	1000	1000
A number of agents are running and have allocated memory.	1000	1000
A new agent is started and uses 100 pages of memory.	1100	1100
A agent using 200 pages of memory terminates. (Notice that 100 pages of memory is freed, while 100 pages is kept allocated for future possible use.)	1000	900
A agent using 50 pages of memory terminates. (Notice that 50 pages of memory is freed and 100 extra pages are still allocated, compared to what is being used by the existing agents.)	950	850
A new agent is started and requires 150 pages of memory. (100 of the 150 pages are already allocated and the database manager only needs to allocate 50 additional pages for this agent.)	1000	1000

A value of "-1," will cause this parameter to use the value of the *min_priv_mem* parameter.

Recommendation: When setting this parameter, you should consider the client connection/disconnection patterns as well as the memory requirements of other processes on the same machine.

If there is only a brief period during which many clients are concurrently connected to the database, a high threshold will prevent unused memory from being decommitted and made available to other processes. This case results in poor memory management which can affect other processes which require memory.

If the number of concurrent clients is more uniform and there are frequent fluctuations in this number, a high threshold will help to ensure memory is available for the client processes and reduce the overhead to allocate and deallocate memory.

Agent/Application Communication Memory

The following parameters affect the amount of memory that is allocated to allow data to be passed between your application and agent processes:

- "Application Support Layer Heap Size (aslheapsz)" on page 492
- "Client I/O Block Size (rqrioblk)" on page 493
- "DOS Requester I/O Block Size (dos_rqrioblk)" on page 494

See "How DB2 Uses Memory" on page 395 for information about how this agent/application shared memory relates to the rest of the memory allocated by the database manager.

Application Support Layer Heap Size (aslheapsz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type	Configurable
Default [Range]	15 [1 – 524 288]
Unit of Measure	Pages (4KB)
When Allocated	When the database manager agent process is started for the local application
When Freed	When the database manager agent process is terminated
Related Parameters	“Query Heap Size (query_heap_sz)” on page 485

The application support layer heap represents a communication buffer between the local application and its associated agent. This buffer is allocated as shared memory by each database manager agent that is started.

If the request to the database manager, or its associated reply, do not fit into the buffer they will be split into two or more send-and-receive pairs. The size of this buffer should be set to handle the majority of requests using a single send-and-receive pair. The size of the request is based on the storage required to hold:

- The input SQLDA
- All of the associated data in the SQLVARs
- The output SQLDA
- Other fields which do not generally exceed 250 bytes.

In addition to this communication buffer, this parameter is also used to determine the I/O block size when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

The data sent from the local application is received by the database manager into a set of contiguous memory allocated from the query heap. The *aslheapsz* parameter is used to determine the initial size of the query heap (for both local and remote clients). The maximum size of the query heap is defined by the *query_heap_sz* parameter.

Recommendation: If your application's requests are generally small and the application is running on a memory constrained system, you may wish to reduce the value of this parameter. If your queries are generally very large, requiring more than one send and receive request, and your system is not constrained by memory, you may wish to increase the value of this parameter.

Use the following formula to calculate the number of pages for *aslheapsz*:

```
aslheapsz >= ( sizeof(input SQLDA)
             + sizeof(each input SQLVAR)
             + sizeof(output SQLDA)
             + 250 ) / 4096
```

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks may also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application. For more information about the OPTIMIZE FOR clause, see "Quickly Retrieving the First Few Rows Using OPTIMIZE FOR n ROWS" on page 289.

Client I/O Block Size (rqrioblk)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 32 767 [4096 – 65 535]

Unit of Measure Bytes

When Allocated

- When a remote client application issues a connection request for a server database
- When a blocking cursor is opened, additional blocks are opened at the client

When Freed

- When the remote application disconnects from the server database
- When the blocking cursor is closed

Related Parameters "DOS Requester I/O Block Size (dos_rqrioblk)" on page 494

This parameter specifies the size of the communication buffer between remote applications and their database agents on the database server. When a database client requests a connection to a remote database, this communication buffer is allocated on the client. On the database server, a communication buffer of 32767 bytes is initially allocated, until a connection is established and the server can determine the value of

rqrioblk at the client. Once the server knows this value, it will reallocate its communication buffer if the client's buffer is not 32767 bytes.

In addition to this communication buffer, this parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

Recommendation: For non-blocking cursors, a reason for increasing the value of this parameter would be if the data (for example, large object data) to be transmitted by a single SQL statement is so large that the default value is insufficient.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks may also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application.

For more information on the OPTIMIZE FOR clause, see “Quickly Retrieving the First Few Rows Using OPTIMIZE FOR n ROWS” on page 289.

DOS Requester I/O Block Size (dos_rqrioblk)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 4096 [4096 – 65 535]

Unit of Measurement Bytes

When Allocated

- When a remote DOS or Windows 3.1 client issues a connection request to a server database
- When a blocking cursor is opened, additional blocks are opened at the client

When Freed

- When the remote application disconnects from the database

- When a blocking cursor is closed

Related Parameters “Client I/O Block Size (rqrioblk)” on page 493

This parameter specifies the size of the communication buffer between DOS/Windows applications and their database agents on the database server. This parameter is similar to the *rqrioblk* parameter, except it allows you to set a different value for blocks used with DOS/Windows clients. In a DB2 configuration file, you can set both the *rqrioblk* parameter (used for OS/2 clients) and the *dos_rqrioblk* parameter (used for DOS clients).

In addition to this communication buffer, this parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

Recommendation: For non-blocking cursors, a reason for increasing the value of this parameter would be if the data (for example, large object data) to be transmitted by a single SQL statement is so large that the default value is insufficient.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks may also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application.

For more information on the OPTIMIZE FOR clause, see “Quickly Retrieving the First Few Rows Using OPTIMIZE FOR n ROWS” on page 289.

Database Manager Instance Memory

The following parameters affect memory that is allocated and used at an instance level:

- “Database System Monitor Heap Size (mon_heap_sz)”
- “Directory Cache Support (dir_cache)” on page 497
- “Maximum Java Interpreter Heap Size (java_heap_sz)” on page 498

Database System Monitor Heap Size (mon_heap_sz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type	Configurable
Default [Range]	
	UNIX 48 [0 – 60 000]
	OS/2 and NT Database Server with local and remote clients 24 [0 – 60 000]
	OS/2 and NT Database Server with local clients 12 [0 – 60 000]
Unit of Measure	Pages (4KB)
When Allocated	When the database manager is started with the *db2start* command
When Freed	When the database manager is stopped with the *db2stop* command
Related Parameters	"Default Database System Monitor Switches (dft_monswitches)" on page 564

This parameter determines the amount of the memory, in pages, to allocate for database system monitor data. Memory is allocated from the monitor heap when you perform database monitoring activities such as taking a snapshot, turning on a monitor switch, resetting a monitor, or activating an event monitor.

A value of zero prevents the database manager from collecting database system monitor data.

Recommendation: The amount of memory required for monitoring activity depends on the number of monitoring applications (applications taking snapshots or event monitors), which switches are set, and the level of database activity.

The following formula provides an approximation of the number of pages required for the monitor heap:

```
( number of monitoring applications + 1 ) *
( number of databases *
  (800 + ( number of tables accessed * 20 )
   + ( ( number of applications connected + 1) *
       (200 + (number of table spaces * 100) ) ) ) )
/ 4096
```

If the available memory in this heap runs out, one of the following will occur:

- A level 2 error message is written to the DB2ALERT.LOG and DB2DIAG.LOG files, when the first application connects to the database for which this event monitor is defined.
- An error code is returned to your application, if an event monitor being started dynamically using the SET EVENT MONITOR statement fails
- An error code is returned to your application, if a monitor command or API subroutine fails

Directory Cache Support (dir_cache)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] Yes [Yes; No]

When Allocated

- When an application issues its first connect, the private cache is allocated
- When a database manager instance is started (`db2start`), the shared cache is allocated.

When Freed

- When an the application process terminates, the private cache is freed
- When a database manager instance is stopped (`db2stop`), the shared cache is freed.

By setting *dir_cache* to “yes” the database, node and DCS directory files will be cached in memory. The use of the directory cache reduces connect costs by eliminating directory file I/O and minimizing the directory searches required to retrieve directory information. There are two types of directory caches:

- A private cache that is allocated and used for each application process, on the machine at which the application is running.
- A shared cache that is allocated and used for some of the internal database manager processes.

Note: Only the private cache is applicable to Windows, Windows 95, Windows NT, and Macintosh environments.

For private caches, when an application issues its first connect, each directory file is read and the information is cached in private memory for this application. The cache is used by the application process on subsequent connect requests and is maintained for the life of the application process. If a database is not found in the private cache, the directory files are searched for the information, but the cache is not updated. If the application modifies a directory entry, the next connect within that application will cause the cache for this application to be refreshed. The private cache for other applications will not be refreshed. When the application process terminates, the cache is freed. (To refresh the directory cache used by a command line processor session, issue a `db2 terminate` command.)

For shared caches, when a database manager instance is started (`db2start`), each directory file is read and the information is cached in shared memory. This cache is

used by some of the database manager processes and is maintained until the instance is stopped (db2stop). If a directory entry is not found in this cache, the directory files are searched for the information. This shared cache is never refreshed during the time the instance is running.

Recommendation: Use directory caching if your directory files do not change frequently and performance is critical.

In addition, on remote clients, directory caching can be beneficial if your applications issue several different connection requests. In this case, caching reduces the number of times a single application must read the directory files.

Directory caching can also improve the performance of taking database system monitor snapshots. In addition, you should explicitly reference the database name on the snapshot call, instead of using database aliases.

Note: Errors may occur when performing snapshot calls if directory caching is turned on and if databases are cataloged, uncataloged, created, or dropped after the database manager is started.

Maximum Java Interpreter Heap Size (java_heap_sz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 512 [0 - 4096]

Unit of Measure Pages (4KB)

When Allocated When a Java application starts

When Freed When a Java application completes

Related Parameters "Java Development Kit 1.1 Installation Path (jdk11_path)" on page 569

This parameter determines the maximum size of the heap that is used by the Java interpreter. For non-partitioned database systems, one heap is allocated for the instance; for partitioned database systems, one heap is allocated for each database partition server.

There is one heap for each DB2 process (one for each agent or subagent on UNIX-based platforms, and one for each instance in other platforms), and there is also one heap for each fenced UDF and fenced stored procedure process. In all situations, only the agents or processes that run Java UDFs or stored procedures ever allocate this memory. On partitioned database systems, the heap is multiplied by the number of database partition servers.

Locks

The following parameters influence how locking is managed in your environment:

- "Time Interval for Checking Deadlock (dlchktime)"
- "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 500
- "Lock Timeout (locktimeout)" on page 501

See also "Maximum Storage for Lock List (locklist)" on page 477.

"Locking" on page 270 provides a general overview of how the database manager uses locking to maintain data integrity.

Time Interval for Checking Deadlock (dlchktime)

Configuration Type Database

Parameter Type Configurable

Default [Range] 10 000 (10 seconds) [1000 – 600 000]

Unit of Measure Milliseconds

Related Parameters

- "Maximum Storage for Lock List (locklist)" on page 477
- "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 500

A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting is never resolved because each application is holding a resource that the other needs to continue.

The deadlock check interval defines the frequency at which the database manager checks for deadlocks among all the applications connected to a database.

Notes:

1. In a partitioned database environment, this parameter applies to the catalog node only.
2. In a partitioned database environment, a deadlock is not flagged until after the second iteration.

Recommendation: Increasing this parameter decreases the frequency of checking for deadlocks, thereby increasing the time that application programs must wait for the deadlock to be resolved.

Decreasing this parameter increases the frequency of checking for deadlocks, thereby decreasing the time that application programs must wait for the deadlock to be resolved but increasing the time that the database manager takes to check for deadlocks. If the deadlock interval is too small, it can decrease run-time performance, because the database manager is frequently performing deadlock detection. If this parameter is set lower to improve concurrency, you should ensure that *maxlocks* and *locklist* are set

appropriately to avoid unnecessary lock escalation, which can result more lock contention and as a result, more deadlock situations.

Maximum Percent of Lock List Before Escalation (maxlocks)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX	10 [1 – 100]
OS/2 and NT	22 [1 – 100]

Unit of Measure Percentage

Related Parameters

- "Maximum Storage for Lock List (locklist)" on page 477
- "Maximum Number of Active Applications (maxappls)" on page 508

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs escalation. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the *maxlocks* value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of *maxlocks*. The *maxlocks* parameter multiplied by the *maxappls* parameter cannot be less than 100.

Recommendation: When setting *maxlocks*, you should consider the size of the lock list (*locklist*):

```
maxlocks = 100 *
           (512 locks per application
           * 32 bytes per lock
           * 2) / (locklist * 4096 bytes)
```

This sample formula allows any application to hold twice the average number of locks.

You can increase *maxlocks* if few applications run concurrently since there will not be a lot of contention for the lock list space in this situation.

You may use the database system monitor to help you track and tune this configuration parameter.

For more information see the *locks_held_top (maximum number of locks held)* monitor element description in the *System Monitor Guide and Reference*. .

The control of lock escalation through this parameter is important to the optimizer since it uses this parameter to determine access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Lock Timeout (locktimeout)

Configuration Type Database

Parameter Type Configurable

Default [Range] -1 [-1; 0 – 30000]

Unit of Measurement Seconds

Related Parameters

- "Maximum Storage for Lock List (locklist)" on page 477
- "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 500

This parameter specifies the number of seconds that an application will wait to obtain a lock. This helps avoid global deadlocks for applications.

If you set this parameter to -1, lock timeout detection is turned off.

Recommendation: In a transaction processing (OLTP) environment, you can use an initial starting value of 30 seconds. In a query-only environment you could start with a higher value. In both cases, you should use benchmarking techniques to tune this parameter.

The value should be set to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of a user leaving their workstation). You should set it high enough so valid lock requests do not time-out because of peak workloads, during which time, there is more waiting for locks.

You may use the database system monitor to help you track the number of times an application (connection) experienced a lock timeout or that a database detected a timeout situation for all applications that were connected.

High values of the *lock_timeout* monitor element can be caused by:

- Too low a value for this configuration parameter
- An application (transaction) that is holding lock(s) for an extended period. You can use the database system monitor to further investigate these applications.
- A concurrency problem, that could be caused by lock escalations (to the row-level to a table-level). See the *maxlocks* and *locklist* parameters.

For more information on the use of this parameter see "Lock Waits and Timeouts" on page 276.

I/O and Storage

The following parameters can influence I/O and storage costs related to the operation of your database:

- "Changed Pages Threshold (chngpgs_thresh)"
- "Number of Asynchronous Page Cleaners (num_iocleaners)"
- "Number of I/O Servers (num_ioservers)" on page 504
- "Index Sort Flag (indexsort)" on page 504
- "Sequential Detection Flag (seqdetect)" on page 505
- "Default Prefetch Size (dft_prefetch_sz)" on page 505
- "Default Number of SMS Containers (numsegs)" on page 506
- "Default Extent Size of Table Spaces (dft_extent_sz)" on page 506
- "Extended Storage Memory Segment Size (estore_seg_sz)" on page 507
- "Number of Extended Storage Memory Segments (num_estore_segs)" on page 507

Changed Pages Threshold (chngpgs_thresh)

Configuration Type Database

Parameter Type Configurable

Default [Range] 60 [5 – 80]

Unit of Measure Percentage

Related Parameters "Number of Asynchronous Page Cleaners (num_iocleaners)"

Asynchronous page cleaners will write changed pages from the buffer pool (or the buffer pools) to disk before the space in the buffer pool is required by a database agent. This means that the agents will not wait for a changed page to be written out, before being able to read a page, and your application's transactions should run faster.

You may use this parameter to specify the level (percentage) of changed pages at which the asynchronous page cleaners will be started, if they are not currently active. When the page cleaners are started, they will build a list of the pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start.

In a read-only (for example, query) environment, these page cleaners are not used.

Recommendation: For databases with a heavy update transaction workload, you can generally ensure that there are enough clean pages in the buffer pool by setting the parameter value to be equal-to or less-than the default value. A percentage larger than the default can help performance if your database has a small number of very large tables.

Number of Asynchronous Page Cleaners (num_iocleaners)

Configuration Type Database

Parameter Type Configurable

Default [Range] 1 [0 – 255]

Unit of Measure Counter

Related Parameters

- “Buffer Pool Size (buffpage)” on page 470
- “Changed Pages Threshold (chngpgs_thresh)” on page 502

This parameter allows you to specify the number of asynchronous page cleaners for a database. These page cleaners write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. This means that the agents will not wait for changed pages to be written out, before being able to read a page. As a result, your application's transactions should run faster.

If you set the parameter to zero (0), no page cleaners are started and as a result, the database agents will perform all of the page writes from the buffer pool to disk. This parameter can have a significant performance impact on a database stored across many physical storage devices, since in this case there is a greater chance that one of the devices will be idle. If no page cleaners are configured, your applications may encounter periodic log full conditions.

If the applications for a database primarily consist of transactions that update data, an increase in the number of cleaners will speed up performance. Increasing the page cleaners will also decrease recovery time from soft failures, such as power outages, because the contents of the database on disk will be more up-to-date at any given time.

Recommendation: Consider the following factors when setting the value for this parameter:

- Application type
  - If it is a query-only database that will not have updates, set this parameter to zero
  - If transactions are run against the database, set this parameter to be between one and the number of physical storage devices used for the database.
- Workload

  Environments with high update transaction rates may require more page cleaners to be configured.
- Buffer pool size (*buffpage*)

  Environments with a large buffer pool may also require more page cleaners to be configured.

You may use the database system monitor to help you tune this configuration parameter using information from the event monitor about write activity from a buffer pool:

- The parameter can be reduced if both of the following conditions are true:
  - `pool_data_writes` is approximately equal to `pool_async_data_writes`
  - `pool_index_writes` is approximately equal to `pool_async_index_writes`.
- The parameter should be increased if either of the following conditions are true:

- `pool_data_writes` is much greater than `pool_async_data_writes`
- `pool_index_writes` is much greater than `pool_async_index_writes`.

For more information see the following monitor elements descriptions in the *System Monitor Guide and Reference*:

- *pool_data_writes (buffer pool data writes)*
- *pool_index_writes (buffer pool index writes)*
- *pool_async_data_writes (buffer pool asynchronous data writes)*
- *pool_async_index_writes (buffer pool asynchronous index writes)*

Number of I/O Servers (num_ioservers)

Configuration Type Database

Parameter Type Configurable

Default [Range] 3 [1 – 255]

Unit of Measure Counter

When Allocated When an application connects to a database

When Freed When an application disconnects from a database

Related Parameters

- "Default Prefetch Size (dft_prefetch_sz)" on page 505
- "Sequential Detection Flag (seqdetect)" on page 505

I/O servers are used on behalf of the database agents to perform prefetch I/O and asynchronous I/O by utilities such as backup and restore. This parameter specifies the number of I/O servers for a database. No more than this number of I/Os for prefetching and utilities can be in progress for a database at any time. An I/O server waits while an I/O operation that it initiated is in progress. Non-prefetch I/Os are scheduled directly from the database agents and as a result are not constrained by *num_ioservers*.

Recommendation: In order to fully exploit all the I/O devices in the system, a good value to use is generally one or two more than the number of physical devices on which the database resides. It is better to configure additional I/O servers, since there is minimal overhead associated with each I/O server and any unused I/O servers will remain remain idle.

For more information, see "Prefetching Data into the Buffer Pool" on page 405 and "Configuring I/O Servers for Prefetching and Parallel I/O" on page 408.

Index Sort Flag (indexsort)

Configuration Type Database

Parameter Type Configurable

Default [Range] Yes [Yes; No]

This parameter indicates whether sorting of index keys will occur during index creation. Performance of index creation is enhanced by performing a sort first, particularly for

indexes with low cluster ratios or cluster factors. Performance of queries can also be better if indexes are created with a sort. The cost of this performance enhancement is the increased disk space required for the sort, which could require twice the amount of space as creating an index without performing an initial sort.

Recommendation: Use the default setting ("Yes"), unless you do not have enough disk space. Note that the disk space required for this sort is approximately equal to the amount of space needed to SELECT the columns of the index from the table with an ORDER BY clause on those columns.

If you have a symmetric multiprocessor (SMP) environment and specify "No" for this parameter, the multiple processing that is possible in an SMP environment is not used during index creation.

Sequential Detection Flag (seqdetect)

Configuration Type Database

Parameter Type Configurable

Default [Range] Yes [Yes; No]

Related Parameters "Default Prefetch Size (dft_prefetch_sz)"

The database manager can monitor I/O and if sequential page reading is occurring the database manager can activate I/O prefetching. This type of sequential prefetch is known as *sequential detection*. You may use the *seqdetect* configuration parameter to control whether the database manager should perform sequential detection.

If this parameter is set to "no," prefetching takes place only if the database manager knows it will be useful, for example table sorts, table scans, or list prefetch.

Recommendation: In most cases, you should use the default value for this parameter. Try turning sequential detection off, only if other tuning efforts were unable to correct serious query performance problems.

Default Prefetch Size (dft_prefetch_sz)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX	32 [0 – 32 767]
OS/2 and NT	16 [0 – 32 767]

Unit of Measure Pages (4KB)

Related Parameters

- "Default Extent Size of Table Spaces (dft_extent_sz)" on page 506
- "Number of I/O Servers (num_ioservers)" on page 504

When a table space is created, `PREFETCHSIZE n` can be optionally specified, where `n` is the number of pages the database manager will read if prefetching is being performed. If you do not specify the prefetch size on the CREATE TABLESPACE statement, the database manager uses the value given by this parameter.

For more information, see “Prefetching Data into the Buffer Pool” on page 405.

Recommendation: Using system monitoring tools, you can determine if your CPU is idle while the system is waiting for I/O. Increasing the value of this parameter may help if the table spaces being used do not have a prefetch size defined for them.

This parameter provides the default for the entire database, and it may not be suitable for all table spaces within the database. For example, a value of 32 may be suitable for a table space with an extent size of 32 pages, but not suitable for a table space with an extent size of 25 pages. Ideally, you should explicitly set the prefetch size for each table space.

To help minimize I/O for table spaces defined with the default extent size (*dft_extent_sz*), you should set this parameter as a factor or whole multiple of the value of the *dft_extent_sz* parameter. For example, if the *dft_extent_sz* parameter is 32, you could set *dft_prefetch_sz* to 16 (a factor of 32) or to 64 (a whole multiple of 32). If the prefetch size is a multiple of the extent size, the database manager may perform I/O in parallel, if the following conditions are true:

- The extents being prefetched are on different physical devices
- Multiple I/O servers are configured (*num_ioservers*).

Default Number of SMS Containers (numsegs)

Configuration Type Database

Parameter Type Informational

Unit of Measure Counter

This parameter, which only applies to SMS table spaces, indicates the number of containers that will be created within the default table spaces. This parameter will show the information used when you created your database, whether it was specified explicitly or implicitly on the CREATE DATABASE command. The CREATE TABLESPACE statement **does not** use this parameter in any way.

For more information, see “Database Physical Directories” on page 25.

Default Extent Size of Table Spaces (dft_extent_sz)

Configuration Type Database

Parameter Type Configurable

Default [Range] 32 [2 – 256]

Unit of Measure Pages (4KB)

Related Parameters “Default Prefetch Size (dft_prefetch_sz)” on page 505

When a table space is created, `EXTENTSIZE n` can be optionally specified, where `n` is the extent size. If you do not specify the extent size on the CREATE TABLESPACE statement, the database manager uses the value given by this parameter.

For more information see “Designing and Choosing Table Spaces” on page 38.

Recommendation: In many cases, you will want to explicitly specify the extent size when you create the table space. Before choosing a value for this parameter, you should understand how you would explicitly choose an extent size for the CREATE TABLESPACE statement. For more information see “Table Space Impact on Query Optimization” on page 304.

Extended Storage Memory Segment Size (estore_seg_sz)

Configuration Type Database

Parameter Type Configurable

Default [Range] 16000 [0 – 1048575]

Unit of Measure Pages (4KB)

Related Parameters “Number of Extended Storage Memory Segments (num_estore_segs)”

This parameter specifies the number of pages in each of the extended memory segments in the database. There are platform-dependent considerations when setting this configuration parameter.

Recommendation: This parameter only has an effect when extended storage is available, and is used as shown by the *num_estore_segs* parameter. When specifying the number of pages to be used in each extended memory segment, you should also consider the number of extended memory segments by reviewing and modifying the *num_estore_segs* parameter. For more information about extended storage, see “Extending Memory” on page 421.

Number of Extended Storage Memory Segments (num_estore_segs)

Configuration Type Database

Parameter Type Configurable

Default [Range] 0 [0 – 2147483647]

Related Parameters “Extended Storage Memory Segment Size (estore_seg_sz)”

This parameter specifies the number of extended storage memory segments available for use by the database.

The default is no extended storage memory segments.

Recommendation: Only use this parameter to establish the use of extended storage memory segments if your platform environment has more memory than the maximum

address space and you wish to use this memory. When specifying the number of segments, you should also consider the size of the each of the segments by reviewing and modifying the *estore_seg_sz* parameter.

When both the *num_estore_segs* and *estore_seg_sz* configuration parameters are set, you should specify which bufferpools will use the extended memory through the CREATE/ALTER BUFFERPOOL statements. For more information about extended storage, see "Extending Memory" on page 421.

Agents

The following parameters can influence the number of applications that can be run concurrently and achieve optimal performance:

- "Maximum Number of Active Applications (maxappls)"
- "Average Number of Active Applications (avg_appls)" on page 509
- "Maximum Database Files Open per Application (maxfilop)" on page 510
- "Maximum Total Files Open per Application (maxtotfilop)" on page 511
- "Priority of Agents (agentpri)" on page 512
- "Maximum Number of Agents (maxagents)" on page 514
- "Maximum Number of Concurrent Agents (maxcagents)" on page 513
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 515
- "Agent Pool Size (num_poolagents)" on page 515
- "Initial Number of Agents in Pool (num_initagents)" on page 516

Maximum Number of Active Applications (maxappls)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX 40 [1 – 5000]

OS/2 and NT Database Server with local and remote clients
20 [1 – 5000]

OS/2 and NT Database Server with local clients
10 [1 – 5000]

Unit of Measure Counter

Related Parameters

- "Maximum Number of Agents (maxagents)" on page 514
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 515
- "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 500
- "Maximum Storage for Lock List (locklist)" on page 477
- "Average Number of Active Applications (avg_appls)" on page 509

This parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

It is important to note that the value of the *maxappls* configuration parameter must be large enough to accommodate the application (user) connections to the database, as well as the number of indoubt units of work and the number of user transactions that have entered, but not yet completed, a two-phase syncpoint process.

When an application attempts to connect to a database, but *maxappls* has already been reached, an error is returned to your application indicating that the maximum number of applications have been connected to your database.

In a partitioned database environment, this is the maximum number of applications that can be concurrently active against a database partition. This parameter limits the number of active applications against the database partition on a database partition server, regardless of whether the server is the coordinator node for the application or not. The catalog node in a partitioned database environment requires a higher value for *maxappls* than is the case for other types of environments because, in the partitioned database environment, every application requires a connection to the catalog node.

Recommendation: Increasing the value of this parameter without lowering the *maxlocks* parameter or increasing the *locklist* parameter could cause you to reach the database limit on locks (*locklist*) rather than the application limit and as a result cause pervasive lock escalation problems.

To a certain extent, the maximum number of applications is also governed by "Maximum Number of Agents (maxagents)" on page 514. An application can only connect to the database, if there is an available connection (*maxappls*) as well as an available agent (*maxagents*). In addition, the maximum number of applications is also controlled by the *max_coordagents* configuration parameter, because no new applications (that is, coordinator agents) can be started if *max_coordagents* has been reached.

Average Number of Active Applications (avg_appls)

Configuration Type Database

Parameter Type Configurable

Default [Range] 1 [1 – maxappls]

Unit of Measure Counter

Related Parameters

- "Maximum Number of Active Applications (maxappls)" on page 508

This parameter is used by the SQL optimizer to help estimate how much buffer pool will be available at run-time for the access plan chosen. Increasing this parameter can

influence the optimizer to choose an access plan for queries that will be more conservative in its buffer pool usage.

Recommendation: When running DB2 in a multi-user environment, particularly with complex queries and a large buffer pool, you may want the SQL optimizer to know that multiple query users are using your system so that the optimizer should be more conservative in assumptions of buffer pool availability.

When setting this parameter, you should estimate the number of heavy query applications that typically use the database. This estimate should exclude all light OLTP applications. If you have trouble estimating this number, you can multiply the following:

- An average number of all applications running against your database. The database system monitor can provide information about the number of applications at any given time and using a sampling technique, you can calculate an average over a period of time. The information from the database system monitor includes both OLTP and non-OLTP applications.
- Your estimate of the percentage of heavy query applications.

As with adjusting other configuration parameters that affect the optimizer, you should adjust this parameter in small increments. This allows you to minimize path selection differences.

You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Maximum Database Files Open per Application (maxfilop)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX	64 [2 – 1950]
OS/2 and NT	64 [2 – 32 768]

Unit of Measure Counter

Related Parameters

- “Maximum Total Files Open per Application (maxtotfilop)” on page 511
- “Maximum Number of Active Applications (maxappls)” on page 508

This parameter specifies the maximum number of file handles that can be open for each database agent. If opening a file causes this value to be exceeded, some files in use by this agent are closed. If *maxfilop* is too small, the overhead of opening and closing files so as not to exceed this limit will become excessive and may degrade performance.

Both SMS table space files and DMS table space containers are treated as files in the database manager's interaction with the operating system, and file handles are required. More files are generally used by SMS table spaces compared to the number of containers used for a DMS table space. Therefore, if you are using SMS table spaces, you will need a larger value for this parameter compared to what you would require for DMS table spaces.

You can also use this parameter to ensure that the overall total of file handles used by the database manager does not exceed the operating system limit by limiting the number of handles per agent to a specific number; the actual number will vary depending on the number of agents running concurrently.

Maximum Total Files Open per Application (maxtotfilop)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 16 000 [100 – 32 768]

Unit of Measure Counter

Related Parameters “Maximum Database Files Open per Application (maxfilop)” on page 510

This parameter defines the maximum number of files that can be opened by all agents and other threads executing in a single database manager instance. If opening a file causes this value to be exceeded, an error is returned to your application.

Note: This parameter does not apply to UNIX-based platforms.

Recommendation: When setting this parameter, you should consider the number of file handles that could be used for each database in the database manager instance. To estimate an upper limit for this parameter:

1. Calculate the maximum number of file handles that could be opened for each database in the instance, using the following formula:

   ```
   maxappls * maxfilop
   ```

2. Calculate the sum of above results and verify that it does not exceed the parameter maximum.

If a new database is created, you should re-evaluate the value for this parameter.

You should also validate the total file handles that may be used on your system does not exceed the system maximum using the following formula:

```
  (sum of maxtotfilop for all instances on machine)
+ (estimate of file handles required by other applications)
<= 65535
```

Priority of Agents (agentpri)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range]

AIX -1 [41 - 125]

Other UNIX
-1 [41 - 128]

Windows NT
-1 [0 - 6]

OS/2 -1 [200 - 231; 300 - 331; 400 - 431]

This parameter controls the priority given to agent and other database manager instance processes and threads by the operating system scheduler. In a partitioned database environment, this also includes both coordinating and parallel agents, the parallel system controllers, and the FCM daemons. This priority determines how CPU time is given to the DB2 processes, agents, and threads relative to the other processes and threads running on the machine. When the parameter is set to -1, no special action is taken and the database manager is scheduled in the normal way that the operating system schedules all processes and threads. When the parameter is set to a value other than -1, the database manager will create its processes and threads with a static priority set to the value of the parameter. Therefore, this parameter allows you to control the priority with which the database manager processes and threads will execute on your machine.

You can use this parameter to increase database manager throughput. The values for setting this parameter are dependent on the operating system on which the database manager is running. For example, in a UNIX-based environment, numerically low values yield high priorities. When the parameter is set to a value between 41 and 125, the database manager creates its agents with a UNIX static priority set to the value of the parameter. This is important in UNIX-based environments because numerically low values yield high priorities for the database manager, but other processes (including applications and users) may experience delays because they cannot obtain enough CPU time. You should balance the setting of this parameter with the other activity expected on the machine.

In an OS/2 environment, higher numeric values yield higher priorities.

For more guidance on using priorities in your operating environment, see the *Quick Beginnings* book for your platform.

Recommendation: The default value should be used initially. This value provides a good compromise between response time to other users/applications and database manager throughput.

If database performance is a concern, you can use benchmarking techniques to determine the optimum setting for this parameter. You should take care when increasing the priority of the database manager because performance of other user processes can be severely degraded especially when the CPU utilization is very high. Increasing the priority of the database manager processes and threads can have significant performance benefits.

Maximum Number of Concurrent Agents (maxcagents)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (*max_coordagents*) [-1; 1 – *max_coordagents*]

Unit of Measure Counter

Related Parameters

- “Maximum Number of Active Applications (maxappls)” on page 508
- “Maximum Number of Agents (maxagents)” on page 514
- “Maximum Number of Coordinating Agents (max_coordagents)” on page 515

The maximum number of database manager coordinator agents that can be concurrently executing a database manager transaction. This parameter is used to control the load on the system during periods of high simultaneous application activity. For example, you may have a system requiring a large number of connections but with a limited amount of memory to serve those connections. Adjusting this parameter can be useful in such an environment, where a period of high simultaneous activity could cause excessive operating system paging.

This parameter does not limit the number of applications that can have connections to a database. It only limits the number of database manager agents that can be processed concurrently by the database manager at any one time, thereby limiting the usage of system resources during times of peak processing.

A value of –1 indicates that the limit is *max_coordagents*.

Recommendation: In most cases the default value for this parameter will be acceptable. In cases where the high concurrency of applications is causing problems, you can use benchmark testing to tune this parameter to optimize your performance.

Maximum Number of Agents (maxagents)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 200 [1 – 64 000]

400 [1 – 64 000] on Partitioned Database Server with local and remote clients

Unit of Measure Counter

Related Parameters

- “Maximum Number of Active Applications (maxappls)” on page 508
- “Maximum Number of Concurrent Agents (maxcagents)” on page 513
- “Maximum Number of Coordinating Agents (max_coordagents)” on page 515
- “Maximum Number of DARI Processes (maxdari)” on page 518
- “Minimum Committed Private Memory (min_priv_mem)” on page 489
- “Agent Pool Size (num_poolagents)” on page 515

This parameter indicates the maximum number of database manager agents, whether coordinating agents or subagents, available at any given time to accept application requests. If you want to limit the number of coordinating agents, use the *max_coordagents* parameter.

This parameter can be useful in memory constrained environments to limit the total memory usage of the database manager, because each additional agent requires additional memory.

Recommendation: The value of *maxagents* should be at least the sum of the values for *maxappls* in each database allowed to be accessed concurrently. If the number of databases is greater than the *numdb* parameter, then the safest course is to use the product of *numdb* with the largest value for *maxappls*.

Each additional agent requires some resource overhead that is allocated at the time the database manager is started.

Maximum Number of Coordinating Agents (max_coordagents)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (*maxagents - num_initagents*) [-1, 0–*maxagents*]

For partitioned database environments and environments in which *parallel_enable* is set to "Yes," the default is *maxagents - num_initagents*; otherwise, the default is *maxagents*. This ensures that, in non-partitioned database environments, *max_coordagents* always equals *maxagents*, unless the system is configured for intra-partition parallelism.

If you do not have a partitioned database environment, and have not enabled the *parallel_enable* parameter, *max_coordagents* must equal *maxagents*.

Related Parameters

- "Initial Number of Agents in Pool (num_initagents)" on page 516
- "Agent Pool Size (num_poolagents)"
- "Maximum Number of Agents (maxagents)" on page 514
- "Enable Intra-Partition Parallelism (intra_parallel)" on page 560

This parameter determines the maximum number of coordinating agents that can exist at one time on a server in a partitioned or non-partitioned database environment.

One coordinating agent is acquired for each local or remote application that connects to a database or attaches to an instance. Requests that require an instance attachment include CREATE DATABASE, DROP DATABASE, and Database System Monitor commands.

Agent Pool Size (num_poolagents)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 [-1, 0-*maxagents*]

Using the default, the value for a server with a non-partitioned database and local clients is the larger of *maxagents*/50 or *max_querydegree*.

Using the default, the value for a server with a non-partitioned database and local and remote clients is the larger of *maxagents*/50 x *max_querydegree* or *maxagents* - *max_coordagents*.

Using the default, the value for an database partition server is the larger of *maxagents*/10 x *max_querydegree* or *maxagents* - *max_coordagents*.

Related Parameters

- "Initial Number of Agents in Pool (num_initagents)"
- "Maximum Number of Agents (maxagents)" on page 514
- "Maximum Query Degree of Parallelism (max_querydegree)" on page 559
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 515

This parameter is a guideline for how large you want the agent pool to grow (and replaces the *max_idleagents* parameter that was used in DB2 Version 2).

The agent pool contains subagents and idle agents. Idle agents can be used as parallel subagents or as coordinating agents. If more agents are created than is indicated by the value of this parameter, they will be terminated when they finish executing their current request, rather than be returned to the pool.

If the value for this parameter is 0, agents will be created as needed, and may be terminated when they finish executing their current request. If the value is *maxagents*, and the pool is full of associated subagents, the server cannot be used as a coordinator node, because no new coordinating agents can be created.

Recommendation: If you run a decision-support environment in which few applications connect concurrently, set *num_poolagents* to a small value to avoid having an agent pool that is full of idle agents.

If you run a transaction-processing environment in which many applications are concurrently connected, increase the value of *num_poolagents* to avoid the costs associated with the frequent creation and termination of agents.

Initial Number of Agents in Pool (num_initagents)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 0 [0–*num_poolagents*]

Related Parameters

- "Maximum Number of Agents (maxagents)" on page 514
- "Agent Pool Size (num_poolagents)" on page 515
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 515

This parameter determines the initial number of idle agents that are created in the agent pool at DB2START time.

Database Application Remote Interface (DARI)

The following parameters can affect the Database Application Remote Interface (DARI) applications.

- "Keep DARI Process Indicator (keepdari)"
- "Maximum Number of DARI Processes (maxdari)" on page 518

Keep DARI Process Indicator (keepdari)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] Yes [Yes; No]

Related Parameters "Maximum Number of DARI Processes (maxdari)" on page 518

This parameter indicates whether or not a DARI process is kept after a DARI call is complete. DARI processes are created as separate system entities in order to isolate user-written DARI code from the database manager agent process. This parameter is only applicable on database servers.

If *keepdari* is set to *no*, a new DARI process is created and destroyed for each DARI invocation. If *keepdari* is set to *yes*, a DARI process is reused for subsequent DARI calls. When the database manager is stopped, all outstanding DARI processes will be terminated.

Setting this parameter to *yes* will result in additional system resources being consumed by the database manager for each DARI process that is activated, up to the value contained in the *maxdari* parameter. This is only true when no existing DARI process is available to process a subsequent DARI call. This parameter is ignored if *maxdari* is set to 0.

Recommendation: In an environment in which the number of DARI requests is large relative to the number of non-DARI requests, and system resources are not constrained, then this parameter can be set to *yes*. This will improve the DARI performance by avoiding the initial DARI process creation overhead since an existing DARI process will be used to process the call.

For example, in an OLTP debit-credit banking transaction application, the code to perform each transaction could be performed in a stored procedure which executes in a DARI process. In this application, the main workload is performed out of DARI processes. If this parameter is set to *no*, each transaction incurs the overhead of creating a new DARI process, resulting in a significant performance reduction. If, however, this parameter is set to *yes*, each transaction would try to use an existing DARI process, which would avoid this overhead.

Maximum Number of DARI Processes (maxdari)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (*max_coordagents*) [-1; 0 – *max_coordagents*]

Unit of Measure Counter

Related Parameters

- “Maximum Number of Agents (maxagents)” on page 514
- “Keep DARI Process Indicator (keepdari)” on page 517
- “Maximum Number of Coordinating Agents (max_coordagents)” on page 515

This parameter indicates the maximum number of DARI process that may reside at the database server. Once this limit is reached, no new DARI requests may be invoked. This parameter is only applicable on database servers.

There can be no more that one DARI process active per agent so that the maximum number of DARI process is also dictated by the maximum number of agents (*maxagents*).

There can be no more than one DARI process active per coordinating agent, so the maximum number of DARI processes is also dictated by the maximum number of coordinating agents (*max_coordagents*).

Recommendation: If your environment features the use of the DARI facility within the database manager, then this parameter can be used to ensure that an appropriate number of DARI processes are available to handle the DARI calls made at any one time within the database manager.

If the parameter is set to –1, the maximum number of DARI processes will be the same as the value set in the *max_coordagents* parameter.

If you find that the default value is not appropriate for your environment because an inappropriate amount of system resource is being given to DARI processes which is affecting performance of the database manager, the following may be useful in providing a starting point for tuning this parameter:

```
maxdari = # of applications allowed to make DARI calls at one time
```

If *keepdari* is set to *yes*, then each DARI process that is created will continue to exist and use system resources even after the DARI call has been processed and returned to the agent.

If your environment is tightly constrained and you cannot afford the process resources associated with DARI, you can disable DARI by setting this parameter to zero (0).

Logging and Recovery

Recovering your environment can be very important to prevent the loss of critical data. A number of parameters are available to help you manage your environment and to ensure that you can perform adequate recovery of your data or transactions. These parameters are grouped into the following categories:

- "Database Log Files"
- "Database Log Activity" on page 524
- "Recovery" on page 528
- "Distributed Unit of Work Recovery" on page 533.

Database Log Files

The following parameters provide information about number, size and status of the files used for database logging:

- "Size of Log Files (logfilsiz)"
- "Number of Primary Log Files (logprimary)" on page 521
- "Number of Secondary Log Files (logsecond)" on page 522
- "Change the Database Log Path (newlogpath)" on page 523
- "Location of Log Files (logpath)" on page 524
- "Next Active Log (nextactive)" on page 524
- "Log Head Identification (loghead)" on page 524

Size of Log Files (logfilsiz)

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX	1000 [4 – 65 535]
OS/2 and NT	250 [4 – 4 095]

Unit of Measure Pages (4KB)

Related Parameters

- "Number of Primary Log Files (logprimary)" on page 521
- "Number of Secondary Log Files (logsecond)" on page 522
- "Recovery Range and Soft Checkpoint Interval (softmax)" on page 526

This parameter defines the size of each primary and secondary log file. The size of these log files limits the number of log records that can be written to them before they become full and a new log file is required.

The use of primary and secondary log files as well as the action taken when a log file becomes full are dependent on the type of logging that is being performed:

- Circular logging

  A primary log file can be reused when the changes recorded in it have been committed. If the log file size is small and applications have processed a large number of changes to the database without committing the changes, a primary log file can quickly become full. If all primary log files become full, the database manager will allocate secondary log files to hold the new log records.
- Log Retention logging

  When a primary log file is full, the log is archived and a new primary log file is allocated.

Recommendation: You must balance the size of the log files with the number of primary log files:

- The value of the *logfilsiz* should be increased if the database has a large number of update, delete and/or insert transactions running against it which will cause the log file to become full very quickly.

  A log file that is too small can affect system performance because of the overhead of archiving old log files, allocating new log files, and waiting for a usable log file.
- The value of the *logfilsiz* should be reduced if disk space is scarce, since primary logs are preallocated at this size.

  A log file that is too large can reduce your flexibility when managing archived log files and copies of log files, since some media may not be able to hold an entire log file.

If you are using log retention, the current active log file is closed and truncated when the last application disconnects from a database. When the next connection to the database occurs, the next log file is used. Therefore, if you understand the logging requirements of your concurrent applications you may be able to determine a log file size which will not allocate excessive amounts of wasted space.

For more information, see the description of this parameter in "Configuration Parameters for Database Logging" on page 217.

Number of Primary Log Files (logprimary)

Configuration Type Database

Parameter Type Configurable

Default [Range] 3 [2 – 128]

Unit of Measure Counter

When Allocated

- The database is created
- A log is moved to a different location (which occurs when the *logpath* parameter is updated)
- Following a increase in the value of this parameter (*logprimary*), during the next database connection after all users have disconnected
- A log file has been archived and a new log file is allocated (the *logretain* or *userexit* parameter must be enabled).
- If the *logfilsiz* parameter has been changed, the active log files are re-sized during the next database connection after all users have disconnected.

When Freed Not freed unless this parameter decreases. If decreased, unneeded log files are deleted during the next connection to the database.

Related Parameters

- "Size of Log Files (logfilsiz)" on page 519
- "Number of Secondary Log Files (logsecond)" on page 522
- "Log Retain Enable (logretain)" on page 527
- "User Exit Enable (userexit)" on page 528

The primary log files establish a fixed amount of storage allocated to the recovery log files. This parameter allows you to specify the number of primary log files to be preallocated.

Under circular logging, the primary logs are used repeatedly in sequence. That is, when a log is full, the next primary log in the sequence is used if it is available. A log is considered available if all units of work with log records in it have been committed or rolled-back. If the next primary log in sequence is not available, then a secondary log is allocated and used. Additional secondary logs are allocated and used until the next primary log in the sequence becomes available or the limit imposed by the *logsecond* parameter is reached. These secondary log files are dynamically deallocated as they are no longer needed by the database manager.

Under log retention, when a log is full, a new log file is used. In this type of logging, log files are never reused. When all units of work with log records in a log file have been committed or rolled-back, the log file can be archived and another primary log is

allocated. If the database manager runs out of pre-allocated primary log files, it will allocate secondary log files as needed.

The number of primary and secondary log files must comply with the following equation:

- (*logprimary* + *logsecond*) <= 128

Recommendation: The value chosen for this parameter depends on a number of factors, including the type of logging being used, the size of the log files, and the type of processing environment (for example, length of transactions and frequency of commits).

Increasing this value will increase the disk requirements for the logs because the primary log files are preallocated during the very first connection to the database.

If you find that secondary log files are frequently being allocated, you may be able to improve system performance by increasing the log file size (*logfilsiz*) or by increasing the number of primary log files.

For databases that are not frequently accessed, in order to save disk storage, set the parameter to 2. For databases enabled for roll-forward recovery, set the parameter larger to avoid the overhead of allocating new logs almost immediately.

You may use the database system monitor to help you size the primary log files.

For more information see the following monitor element descriptions in the *System Monitor Guide and Reference*:

- *sec_log_used_top (maximum secondary log space used)*
- *tot_log_used_top (maximum total log space used)*
- *sec_logs_allocated (secondary logs allocated currently)*

Observation of these monitor values over a period of time will aid in better tuning decisions, as average values may be more representative of your ongoing requirements.

Number of Secondary Log Files (logsecond)

Configuration Type Database

Parameter Type Configurable

Default [Range] 2 [0 – 126]

Unit of Measure Counter

When Allocated As needed when *logprimary* is insufficient (see detail below)

When Freed Over time as the database manager determines they will no longer be required.

Related Parameters

- “Size of Log Files (logfilsiz)” on page 519
- “Number of Primary Log Files (logprimary)” on page 521

- "Log Retain Enable (logretain)" on page 527
- "User Exit Enable (userexit)" on page 528

This parameter specifies the number of secondary log files that are created and used for recovery log files (only as needed). When the primary log files become full, the secondary log files (of size *logfilsiz*) are allocated one at a time as needed, up to a maximum number as controlled by this parameter. An error code will be returned to the application, and the database will be shutdown, if more secondary log files are required than are allowed by this parameter.

See "Number of Primary Log Files (logprimary)" on page 521 for more information about how secondary logs are used.

Recommendation: Use secondary log files for databases that have periodic needs for large amounts of log space. For example, an application that is run once a month may require log space beyond that provided by the primary log files. Since secondary log files do not require permanent file space they are advantageous in this type of situation.

Change the Database Log Path (newlogpath)

Configuration Type Database

Parameter Type Configurable

Default [Range] Null [any valid path]

Related Parameters

- "Location of Log Files (logpath)" on page 524
- "Database is Consistent (database_consistent)" on page 539

This parameter allows you to change the location where log files are stored. The path name that you specify must be a fully qualified path name, not a relative path name, and it cannot exceed 242 bytes.

Note: In a partitioned database environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

This path does not become the value of *logpath* until both of the following occur:

- The database is in a consistent state, as indicated by the *database_consistent* parameter.
- All users are disconnected from the database

When the first new connection is made to the database, the database manager will move the logs to this location.

Recommendation: Ideally, the log files will be on a physical disk which does **not** have high I/O. For instance, avoid putting the logs on the same disk as the operating system or high volume databases. This will allow for efficient logging activity with a minimum of overhead such as waiting for I/O.

You may use the database system monitor to track the number of I/O's related to database logging.

For more information, see the following monitor element descriptions in the *System Monitor Guide and Reference*:

- *log_reads (number of log pages read)*
- *log_writes (number of log pages written)*.

The preceding data elements return the amount of I/O activity related to database logging. You can use an operating system monitor tool to collect information about other disk I/O activity, then compare the two types of I/O activity.

Location of Log Files (logpath)

Configuration Type Database

Parameter Type Informational

Related Parameters “Change the Database Log Path (newlogpath)” on page 523

This parameter contains the current path being used for logging purposes. You cannot change this parameter directly as it is set by the database manager after a change to the *newlogpath* parameter becomes effective.

When a database is created, the recovery log file for it is created in a subdirectory of the directory containing the database. The default is a subdirectory named SQLOGDIR under the directory created for the database.

Next Active Log (nextactive)

Configuration Type Database

Parameter Type Informational

This parameter contains the name of log file that will be used once the current active log is full. When log retention is being used, all log files with a sequence number greater than or equal to this file's sequence number, are not used, although they are preallocated to enhance performance and ensure the space is available when required.

Log Head Identification (loghead)

Configuration Type Database

Parameter Type Informational

This parameter contains the name of the log file that is currently active.

Database Log Activity

The following parameters can influence the type and performance of database logging:

- “Number of Commits to Group (mincommit)” on page 525
- “Recovery Range and Soft Checkpoint Interval (softmax)” on page 526
- “Log Retain Enable (logretain)” on page 527

- “User Exit Enable (userexit)” on page 528

Number of Commits to Group (mincommit)

Configuration Type Database

Parameter Type Configurable

Default [Range] 1 [1 – 25]

Unit of Measure Counter

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed. This delay can help reduce the database manager overhead associated with writing log records and as a result improve performance when you have multiple applications running against a database and many commits are requested by the applications within a very short time frame.

This grouping of commits will only occur when the value of this parameter is greater than one and when the number of applications connected to the database is greater than or equal to the value of this parameter. When commit grouping is being performed, application commit requests are held until either one second has elapsed or the number of commit requests equals the value of this parameter.

Recommendation: Increase this parameter from its default value if multiple read/write applications typically request concurrent database commits. This will result in more efficient logging file I/O as it will occur less frequently and write more log records each time it does occur.

You could also sample the number of transactions per second and adjust this parameter to accommodate the peak number of transactions per second (or some large percentage of it). Accommodating peak activity would minimize the overhead of writing log records during heavy load periods.

If you increase *mincommit*, you may also need to increase the *logbufsz* parameter to avoid having a full log buffer force a write during these heavy load periods. In this case, the *logbufsz* should be equal to:

```
mincommit * (log space used, on average, by a transaction)
```

You may use the database system monitor to help you tune this parameter in the following ways:

- Calculating the peak number of transactions per second:

  Taking monitor samples through-out a typical day, you can determine your heavy load periods. You can calculate the total transactions by adding the following monitor elements:
  - *commit_sql_stmts (commit statements attempted)*
  - *rollback_sql_stmts (rollback statements attempted)*

  Using this information and the available timestamps, you can calculate the number of transactions per second.
- Calculating the log space used per transaction:

Using sampling techniques over a period of time and a number of transactions, you can calculate an average of the log space used with the following monitor element:

– *log_space_used (unit of work log space used)*

For more information about the database system monitor, see the *System Monitor Guide and Reference*.

Recovery Range and Soft Checkpoint Interval (softmax)

Configuration Type Database

Parameter Type Configurable

Default [Range] 100 [1 – 100 * *logprimary*]

Unit of Measure Percentage of total number of primary log files

Related Parameters

- "Size of Log Files (logfilsiz)" on page 519
- "Number of Primary Log Files (logprimary)" on page 521

This parameter is used to:

- Influence the number of logs that need to be recovered following a crash (such as a power failure). For example, if the default value is used, the database manager will try to keep the number of logs that need to be recovered to 1. If you specify 300 as the value of this file, the database manager will try to keep the number of logs that need to be recovered to 3.

  To influence the number of logs required for crash recovery, the database manager uses this parameter to trigger the page cleaners to ensure that pages older than the specified recovery window are already written to disk.

- Determine the frequency of soft checkpoints.

At the time of a database failure resulting from an event such as a power failure, there may have been changes to the database which:

- Have not been committed, but updated the data in the buffer pool
- Have been committed, but have not been written from the buffer pool to the disk
- Have been committed and written from the buffer pool to the disk.

When a database is restarted, the log files will be used to perform a crash recovery of the database which ensures that the database is left in a consistent state (that is, all committed transactions are applied to the database and all uncommitted transactions are not applied to the database).

To determine which records from the log file need to be applied to the database, the database manager uses a log control file. This log control file is periodically written to disk, and, depending on the frequency of this event, the database manager may be applying log records of committed transactions or applying log records that describe changes that have already been written from the buffer pool to disk. These log records have no impact on the database, but applying them introduces some overhead into the database restart process.

The log control file is always written to disk when a log file is full, and during soft checkpoints. You can use this configuration parameter to trigger additional soft checkpoints.

The timing of soft checkpoints is based on the difference between the “current state” and the “recorded state,” given as a percentage of the *logfilsiz*. The “recorded state” is determined by the oldest valid log record indicated in the log control file on disk, while the “current state” is determined by the log control information in memory. (The oldest valid log record is the first log record that the recovery process would read.) The soft checkpoint will be taken if the value calculated by the following formula is greater than or equal to the value of this parameter:

```
( (space between recorded and current states) / logfilsiz ) * 100 * logprimary
```

Recommendation: You may want to increase or reduce the value of this parameter, depending on whether your acceptable recovery window is greater than or less than one log file. Lowering the value of this parameter will cause the database manager both to trigger the page cleaners more often and to take more frequent soft checkpoints. These actions can reduce both the number of log records that need to be processed and the number of redundant log records that are processed during crash recovery.

Note however, that more page cleaner triggers and more frequent soft checkpoints increase the overhead associated with database logging, which can impact the performance of the database manager. Also, more frequent soft checkpoints may not reduce the time required to restart a database, if you have:

- Very long transactions with few commit points.
- A very large buffer pool and the pages containing the committed transactions are not written back to disk very frequently. (Note that the use of asynchronous page cleaners can help avoid this situation. See “Number of Asynchronous Page Cleaners (num_iocleaners)” on page 502.)

In both of these cases, the log control information kept in memory does not change frequently and there is no advantage in writing the log control information to disk, unless it has changed.

Log Retain Enable (logretain)

Configuration Type Database

Parameter Type Configurable

Default [Range] No [Yes; No]

Related Parameters

- “User Exit Enable (userexit)” on page 528
- “Log Retain Status Indicator (log_retain_status)” on page 540
- “Backup Pending Indicator (backup_pending)” on page 539

If either *logretain* or *userexit* are enabled, the active log files will be retained and become online archive log files for use in roll-forward recovery. This is called log retention logging.

After *logretain*, or *userexit*, or both of these parameters are enabled, you must make a full backup of the database. This state is indicated by the *backup_pending* flag parameter.

If both of these parameters are de-selected, roll-forward recovery becomes unavailable for the database because logs will no longer be retained. In this case, the database manager deletes all log files in the *logpath* directory (including online archive log files), allocates new active log files, and reverts to circular logging.

User Exit Enable (userexit)

Configuration Type Database

Parameter Type Configurable

Default [Range] No [Yes; No]

Related Parameters

- "Log Retain Enable (logretain)" on page 527
- "User Exit Status Indicator (user_exit_status)" on page 540
- "Backup Pending Indicator (backup_pending)" on page 539

If this parameter is enabled, log retention logging is performed regardless of how the *logretain* parameter is set. This parameter also indicates that a user exit program should be used to archive and retrieve the log files. Log files are archived when the database manager closes the log file. They are retrieved when the ROLLFORWARD utility needs to use them to restore a database.

After *logretain*, or *userexit*, or both of these parameters are enabled, you must make a full backup of the database. This state is indicated by the *backup_pending* flag parameter.

If both of these parameters are de-selected, roll-forward recovery becomes unavailable for the database because logs will no longer be retained. In this case, the database manager deletes all log files in the *logpath* directory (including online archive log files), allocates new active log files, and reverts to circular logging.

For more information on the user exit program, see Appendix J, "User Exit for Database Recovery" on page 753.

Recovery

The following parameters affect various aspects of database recovery:

- "Auto Restart Enable (autorestart)" on page 529
- "Index Re-creation Time (indexrec)" on page 529

- “Default Number of Load Recovery Sessions (dft_loadrec_ses)” on page 530
- “Recovery History Retention Period (rec_his_retentn)” on page 531

See also “Distributed Unit of Work Recovery” on page 533.

The following parameters are used when working with ADSTAR Distributed Storage Manager (ADSM):

- “ADSTAR Distributed Storage Manager Management Class (adsm_mgmtclass)” on page 531
- “ADSTAR Distributed Storage Manager Password (adsm_password)” on page 532
- “ADSTAR Distributed Storage Manager Node Name (adsm_nodename)” on page 532
- “ADSTAR Distributed Storage Manager Owner Name (adsm_owner)” on page 532

Auto Restart Enable (autorestart)

Configuration Type Database

Parameter Type Configurable

Default [Range] On [On; Off]

When this parameter is set on, the database manager automatically calls the restart database utility, if needed, when an application connects to a database. *Crash recovery* is the operation performed by the restart database utility. It is performed if the database terminated abnormally while applications were connected to it. An abnormal termination of the database could be caused by a power failure or a system software failure. It applies any committed transactions that were in the database buffer pool but were not written to disk at the time of the failure. It also backs out any uncommitted transactions that may have been written to disk.

If *autorestart* is not enabled, then an application that attempts to connect to a database which needs to have crash recovery performed (needs to be restarted) will receive a SQL1015N error. In this case, the application can call the restart database utility, or you can restart the database by selecting the restart operation of the recovery tool.

Index Re-creation Time (indexrec)

Configuration Type Database and Database Manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX Database Mgr. restart [restart; access]

OS/2 and NT Database Mgr.
access [restart; access]

	Database	Use system setting [system; restart; access]

Related Parameters “Auto Restart Enable (autorestart)” on page 529

This parameter indicates when the database manager will attempt to re-build invalid indexes. There are three possible settings for this parameter:

SYSTEM *use system setting* which will cause invalid indexes to be re-built at the time specified in the database manager configuration file. (Note: This setting is only valid for database configurations.) (The API constant for this value is SQLF_INX_REC_SYSTEM)

ACCESS *during index access* which will cause invalid indexes to be re-built when the index is first accessed. (The API constant for this value is SQLF_INX_REC_REFERENCE)

RESTART *during database restart* which will cause invalid indexes to be re-built when a RESTART DATABASE command is either explicitly or implicitly issued. Note that a RESTART DATABASE command is implicitly issued if the *autorestart* parameter is enabled. (The API constant for this value is SQLF_INX_REC_RESTART)

Indexes can become invalid when fatal disk problems occur. If this happens to the data itself, the data could be lost. However, if this happens to an index, the index can be recovered by re-creating it. If an index is re-built while users are connected to the database, two problems could occur:

- An unexpected degradation in response time may occur as the index file is re-created. Users accessing the table and using this particular index would wait while the index was being re-built.
- Unexpected locks may be held after index re-creation, especially if the user transaction that caused the index to be re-created never performed a COMMIT or ROLLBACK.

Recommendation: The best choice for this option on a high-user server and if restart time is not a concern, would be to have the index re-built at DATABASE RESTART time as part of the process of bringing the database back online after a crash.

Setting this parameter to “ACCESS” will result in a degradation of the performance of the database manager while the index is being re-created. Any user accessing that specific index or table would have to wait until the re-creating is complete.

If this parameter is set to “RESTART,” the time taken to restart the database will be longer due to index re-creation but normal processing would not be impacted once the database has been brought back online.

Default Number of Load Recovery Sessions (dft_loadrec_ses)

Configuration Type Database

Parameter Type Configurable

Default [Range] 1 [1 – 30 000]

Unit of Measurement Counter

This parameter specifies the default number of sessions that will be used during the recovery of a table load. The value should be set to an optimal number of I/O sessions to be used to retrieve a load copy. The retrieval of a load copy is an operation similar to restore. You can override this parameter through entries in the copy location file specified by the environment variable DB2LOADREC.

The default number of buffers used for load retrieval is two more than the value of this parameter. You can also override the number of buffers in the copy location file.

This parameter is applicable only if roll forward recovery is enabled.

For more information about load recovery, see "Creating a Copy Image of Loaded Data" on page 151.

Recovery History Retention Period (rec_his_retentn)

Configuration Type Database

Parameter Type Configurable

Default [Range] 366 [-1; 0 – 30 000]

Unit of Measure Days

This parameter is used to specify the number of days that historical information on backups should be retained. If the recovery history file is not needed to keep track of backups, restores, and loads, this parameter can be set to a small number.

If value of this parameter is -1, the recovery history file can only be pruned explicitly using the available commands or APIs. If the value is not -1, the recovery history file is pruned after every full database backup.

No matter how small the retention period, the most recent full database backup plus its restore set will always be kept, unless you use the PRUNE utility with the FORCE option. For more information about this utility, see the *Command Reference*.

ADSTAR Distributed Storage Manager Management Class (adsm_mgmtclass)

Configuration Type Database

Parameter Type Configurable

Default [Range] Null [any string]

The ADSTAR Distributed Storage Manager management class tells how the ADSM server should manage the backup versions of the objects being backed up.

The default is that there is no ADSM management class.

The management class is assigned from the ADSTAR Distributed Storage Manager administrator. Once assigned, you should set this parameter to the management class name. When performing any ADSM backup, the database manager uses this parameter to pass the management class to ADSM.

ADSTAR Distributed Storage Manager Password (adsm_password)

Configuration Type Database

Parameter Type Configurable

Default [Range] Null [any string]

This parameter is used to override the default setting for the password associated with the ADSTAR Distributed Storage Manager (ADSM) product. The password is needed to allow you to restore a database that was backed up to ADSM from another node.

Note: If the *adsm_nodename* is overridden during a backup done with DB2 (for example, with the BACKUP DATABASE command), the *adsm_password* may also have to be set.

The default is that you can only restore a database from ADSM on the same node from which you did the backup. It is possible for the *adsm_nodename* to be overridden during a backup done with DB2.

For more information on ADSTAR Distributed Storage Manager, see “ADSTAR Distributed Storage Manager” on page 230.

ADSTAR Distributed Storage Manager Node Name (adsm_nodename)

Configuration Type Database

Parameter Type Configurable

Default [Range] Null [any string]

This parameter is used to override the default setting for the node name associated with the ADSTAR Distributed Storage Manager (ADSM) product. The node name is needed to allow you to restore a database that was backed up to ADSM from another node.

The default is that you can only restore a database from ADSM on the same node from which you did the backup. It is possible for the *adsm_nodename* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

For more information on ADSTAR Distributed Storage Manager, see “ADSTAR Distributed Storage Manager” on page 230.

ADSTAR Distributed Storage Manager Owner Name (adsm_owner)

Configuration Type Database

Parameter Type Configurable

Default [Range] Null [any string]

This parameter is used to override the default setting for the owner associated with the ADSTAR Distributed Storage Manager (ADSM) product. The owner name is needed to allow you to restore a database that was backed up to ADSM from another node. It is possible for the *adsm_owner* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

Note: The owner name is case sensitive.

The default is that you can only restore a database from ADSM on the same node from which you did the backup.

For more information on ADSTAR Distributed Storage Manager, see "ADSTAR Distributed Storage Manager" on page 230.

Distributed Unit of Work Recovery

The following parameters affect the recovery of Distributed Unit of Work (DUOW) transactions:

- "Transaction Manager Database Name (tm_database)"
- "Transaction Resync Interval (resync_interval)" on page 534
- "Sync Point Manager Name (spm_name)" on page 534
- "Sync Point Manager Log File Size (spm_log_file_sz)" on page 535
- "Sync Point Manager Resync Agent Limit (spm_max_resync)" on page 536

Transaction Manager Database Name (tm_database)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 1ST_CONN [any valid database name]

This parameter identifies name of the Transaction Manager (TM) database for each DB2 instance. A TM database can be a local database or a remote database that is not accessed through DRDA protocols. The TM database is a database that is used as a logger and coordinator, and is used to perform recovery for indoubt transactions.

You may set this parameter to **1ST_CONN** which will set the TM database to be the first database to which a user connects.

For more information, see Chapter 7, "Distributed Databases" on page 239.

Recommendation: For simplified administration and operation you may wish to create a few databases over a number of instances and use these databases exclusively as TM databases.

Transaction Resync Interval (resync_interval)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 180 [60 – 60 000]

Unit of Measurement Seconds

This parameter specifies the time interval in seconds for which a Transaction Manager (TM), Resource Manager (RM) or Sync Point Manager (SPM) should retry the recovery of any outstanding indoubt transactions found in the TM, the RM, or the SPM. This parameter is applicable when you have transactions running in a distributed unit of work (DUOW) environment.

For more information see Chapter 7, “Distributed Databases” on page 239.

Recommendation: If, in your environment, indoubt transactions will not interfere with other transactions against your database, you may wish to increase the value of this parameter. If you are using a DB2 Connect gateway to access DRDA2 Application Servers, you should consider the effect indoubt transactions may have at the Application Servers even though there will be no interference with local data access. If there are no indoubt transactions, the performance impact will be minimal.

Sync Point Manager Name (spm_name)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null [any valid database name]

This parameter identifies the name of the Sync Point Manager (SPM) instance to the database manager. The *spm_name* must be defined in the system database directory and, if remote, in the node directory.

Note: This parameter is not applicable to the Windows NT environment.

For more information on the Sync Point Manager, see the appendix on "LU 6.2 Sync Point Manager Considerations" in the *Quick Beginnings* or the *DB2 Connect Enterprise Edition Quick Beginnings* appropriate to your operating system environment. For more information on recovery of indoubt DRDA transactions, see "Recovery of Indoubt DRDA Transactions" on page 252.

Sync Point Manager Log File Size (spm_log_file_sz)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 256 [4 – 1000]

Unit of Measure Pages (4KB)

This parameter identifies the Sync Point Manager (SPM) log file size in 4K pages. The log file is contained in the `spmlog` sub-directory under `sqllib` and is created the first time SPM is started.

Note: This parameter is not applicable to the Windows NT environment.

For more information on the Sync Point Manager, see the appendix on "LU 6.2 Sync Point Manager Considerations" in the *Quick Beginnings* or the *DB2 Connect Enterprise Edition Quick Beginnings* appropriate to your operating system environment.

For more information on recovery of indoubt DRDA transactions, see "Recovery of Indoubt DRDA Transactions" on page 252.

Recommendation: The Sync Point Manager log file size should be large enough to maintain performance, but small enough to prevent wasted space. The size required depends on the number of transactions using protected conversations, and how often COMMIT or ROLLBACK is issued.

To change the size of the SPM log file:

1. Determine that there are no indoubt transactions by using the LIST DRDA INDOUBT TRANSACTIONS command.
2. If there are none, stop the database manager.
3. Update the Database Manager Configuration with a new SPM log file size.
4. Go to the `$HOME/sqllib` directory and issue `rm -fr spmlog` to delete the current SPM log. (Note: This shows the AIX command. Other systems may require a different remove or delete command.)
5. Start the database manager. (A new SPM log of the specified size is created during the startup of the database manager.)

Sync Point Manager Resync Agent Limit (spm_max_resync)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 20 [10 – 256]

This parameter identifies the number of agents that can simultaneously perform resync operations.

Note: This parameter is not applicable to the Windows NT environment.

For more information on recovery of indoubt DRDA transactions, see “Recovery of Indoubt DRDA Transactions” on page 252. For more information on the Sync Point Manager, see the appendix on “LU 6.2 Sync Point Manager Considerations” in the *Quick Beginnings* or the *DB2 Connect Enterprise Edition Quick Beginnings* appropriate to your operating system environment.

Database Management

A number of parameters are available which provide information about your database or influence the management of your database. These are grouped as follows:

- “Attributes”
- “Status” on page 539
- “Compiler Settings” on page 541.

Attributes

The following parameters provide general information about the database:

- “Configuration File Release Level (release)”
- “Database Release Level (database_level)” on page 537
- “Territory for the Database (territory)” on page 537
- “Country code for the Database (country)” on page 537
- “Codeset for the Database (codeset)” on page 537
- “Code Page for the Database (codepage)” on page 538
- “Collating Information (collate_info)” on page 538
- “Copy Protection Enable (copyprotect)” on page 538

With the exception of *copyprotect*, these parameters are provided for informational purposes only.

Configuration File Release Level (release)

Configuration Type Database manager, Database

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Informational

Related Parameters “Database Release Level (database_level)”

This parameter specifies the release level of the configuration file.

Database Release Level (database_level)

Configuration Type Database

Parameter Type Informational

Related Parameters “Configuration File Release Level (release)” on page 536

This parameter indicates the release level of the database manager which can use the database. In the case of an incomplete or failed migration, this parameter will reflect the release level of the unmigrated database and may differ from the *release* parameter (the release level of the database configuration file). Otherwise the value of *database_level* will be identical to value of the *release* parameter.

Territory for the Database (territory)

Configuration Type Database

Parameter Type Informational

Related Parameters “Country code for the Database (country)”

This parameter shows the territory used to create the database. Territory is used by the database manager to determine *country* parameter values. For more information about how the database manager uses the territory, see the *Quick Beginnings*.

Country code for the Database (country)

Configuration Type Database

Parameter Type Informational

Related Parameters “Territory for the Database (territory)”

This parameter shows the country code used to create the database. The *country* parameter is derived based on the *territory* parameter. For more information, see the *Quick Beginnings*.

Codeset for the Database (codeset)

Configuration Type Database

Parameter Type Informational

Related Parameters “Code Page for the Database (codepage)” on page 538

This parameter shows the codeset that was used to create the database. Codeset is used by the database manager to determine *codepage* parameter values. For more information about how the database manager uses the codeset, see the *Quick Beginnings*.

Code Page for the Database (codepage)

Configuration Type Database

Parameter Type Informational

Related Parameters “Codeset for the Database (codeset)” on page 537

This parameter shows the code page that was used to create the database. The *codepage* parameter is derived based on the *codeset* parameter. For more information, see the *Quick Beginnings*.

Collating Information (collate_info)

This parameter can only be displayed using the GET DATABASE CONFIGURATION API. It **cannot** be displayed through the command line processor or the Control Center.

Configuration Type Database

Parameter Type Informational

This parameter provides 260 bytes of database collating information. The first 256 bytes specify the database collating sequence, where byte “n” contains the sort weight of the code point whose underlying decimal representation is “n” in the code page of the database.

The last 4 bytes contain internal information about the type of the collating sequence. You can treat it as an integer applicable to the platform of the database. There are three values:

- **0** – The sequence contains non-unique weights
- **1** – The sequence contains all unique weights
- **2** – The sequence is the identity sequence, for which strings are compared byte for byte.

If you use this internal type information, you need to consider byte reversal when retrieving information for a database on a different platform.

You can specify the collating sequence at database creation time.

Copy Protection Enable (copyprotect)

Configuration Type Database

Parameter Type Configurable

Default [Range] No [Yes; No]

This parameter enables the copy-protect attribute and is disabled by default. Prior to Version 2 of the database manager, the default was to enable the copy-protect attribute.

This parameter does not apply to UNIX-based environments.

The backup database and restore database utilities are not affected by the *copyprotect* parameter. It is possible to back up a copy-protected database, restore it to a different workstation, and then catalog and access the database.

Attention: Remove copy-protection from all databases before reinstalling either the database manager or the operating system. If you do not remove copy-protection, you will receive an error when you attempt to access the database. After you have reinstalled, you can enable copy-protection.

Status

The following parameters provide information about the state of the database:

- "Backup Pending Indicator (backup_pending)"
- "Database is Consistent (database_consistent)"
- "Roll Forward Pending Indicator (rollfwd_pending)" on page 540
- "Log Retain Status Indicator (log_retain_status)" on page 540
- "User Exit Status Indicator (user_exit_status)" on page 540
- "Restore Pending (restore_pending)" on page 540
- "MultiPage File Allocation Enabled (multipage_alloc)" on page 540

Backup Pending Indicator (backup_pending)

Configuration Type Database

Parameter Type Informational

If set on, this parameter indicates that you must do a full backup of the database before accessing the database. This parameter is turned on after either *logretain* or *userexit* has been set and accepted.

Database is Consistent (database_consistent)

Configuration Type Database

Parameter Type Informational

This parameter indicates whether the database is in a consistent state.

YES indicates that all transactions have been committed or rolled back so that the data is consistent. If the system "crashes" while the database is consistent, you do not need to take any special action to make the database usable.

NO indicates that a transaction is pending or some other task is pending on the database and the data is not consistent at this point. If the system "crashes" while the database is not consistent, you will need to restart the database using the RESTART

DATABASE command to make the database usable. For more information about the RESTART DATABASE command, see the *Command Reference*.

Roll Forward Pending Indicator (rollfwd_pending)

Configuration Type Database

Parameter Type Informational

This parameter can indicate one of the following states:

- **DATABASE**, meaning that a roll-forward recovery procedure is required for this database
- **TABLESPACE**, meaning that one or more table space needs to be rolled forward
- **NO**, meaning that the database is usable and no roll-forward recovery is required.

The recovery (using ROLLFORWARD DATABASE) must complete before you can access the database or table space. For more information about ROLLFORWARD DATABASE, see the *Command Reference*.

Log Retain Status Indicator (log_retain_status)

Configuration Type Database

Parameter Type Informational

Related Parameters "Log Retain Enable (logretain)" on page 527

If set, this parameter indicates that log files are being retained for use in roll-forward recovery.

This parameter is set when the *logretain* parameter setting becomes active.

User Exit Status Indicator (user_exit_status)

Configuration Type Database

Parameter Type Informational

Related Parameters "User Exit Enable (userexit)" on page 528

If set ON, this indicates that the database manager is enabled for roll-forward recovery and that the user exit program will be used to archive and retrieve log files when called by the database manager.

Restore Pending (restore_pending)

Configuration Type Database

Parameter Type Informational

This parameter states whether a RESTORE PENDING status exists in the database.

MultiPage File Allocation Enabled (multipage_alloc)

Configuration Type Database

Parameter Type Informational

Multipage file allocation is used to improve insert performance. It applies to SMS table spaces only. If enabled, all SMS table spaces are affected: there is no selection possible for individual SMS table spaces.

The default for the parameter is NO: multipage file allocation is not enabled.

Following database creation, the parameter may be set to YES which indicates that multipage file allocation is enabled. This is done using the *db2empfa* tool. Once set to YES, the parameter cannot be changed back to NO.

Compiler Settings

The following parameters provide information to influence the compiler:

- "Continue upon Arithmetic Exceptions (dft_sqlmathwarn)"
- "Default Degree (dft_degree)" on page 542
- "Default Query Optimization Class (dft_queryopt)" on page 543
- "Number of Frequent Values Retained (num_freqvalues)" on page 544
- "Number of Quantiles for Columns (num_quantiles)" on page 544

Continue upon Arithmetic Exceptions (dft_sqlmathwarn)

Configuration Type Database

Parameter Type Configurable

Default [Range] No [No, Yes]

This parameter sets the default value that determines the handling of arithmetic errors and retrieval conversion errors as errors or warnings during SQL statement compilation. For static SQL statements, the value of this parameter is associated with the package at bind time. For dynamic SQL DML statements, the value of this parameter is used when the statement is prepared.

Attention: If you change the *dft_sqlmathwarn* value for a database, the behaviour of check constraints, triggers, and views that include arithmetic expressions may change. This may in turn have an impact on the data integrity of the database. You should only change the setting of *dft_sqlmathwarn* for a database after carefully evaluating how the new arithmetic exception handling behaviour may impact check constraints, triggers, and views. Once changed, subsequent changes require the same careful evaluation.

As an example, consider the following check constraint, which includes a division arithmetic operation:

```
A/B > 0
```

When *dft_sqlmathwarn* is "No" and an INSERT with B=0 is attempted, the division by zero is processed as an arithmetic error. The insert operation fails because DB2 cannot check the constraint. If *dft_sqlmathwarn* is changed to "Yes," the division by zero is processed as an arithmetic warning with a NULL result. The NULL result causes the ">" predicate to evaluate to UNKNOWN and the insert operation succeeds. If *dft_sqlmathwarn* is changed back to "No," an attempt to insert the same row will fail, because the division by zero error prevents DB2 from evaluating the constraint. The

row inserted with B=0 when *dft_sqlmathwarn* was “Yes” remains in the table and can be selected. Updates to the row that cause the constraint to be evaluated will fail, while updates to the row that do not require constraint re-evaluation will succeed.

Before changing *dft_sqlmathwarn* from “No” to “Yes,” you should consider rewriting the constraint to explicitly handle nulls from arithmetic expressions. For example:

```
( A/B > 0 ) AND ( CASE
                    WHEN A IS NULL THEN 1
                    WHEN B IS NULL THEN 1
                    WHEN A/B IS NULL THEN 0
                    ELSE 1
                    END
                  = 1 )
```

can be used if both A and B are nullable. And, if A or B is not-nullable, the corresponding IS NULL WHEN-clause can be removed.

Before changing *dft_sqlmathwarn* from “Yes” to “No,” you should first check for data that may become inconsistent, for example by using predicates such as the following:

```
WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

When inconsistent rows are isolated, you should take appropriate action to correct the inconsistency before changing *dft_sqlmathwarn*. You can also manually re-check constraints with arithmetic expressions after the change. To do this, first place the affected tables in a check pending state (with the OFF clause of the SET CONSTRAINTS statement), then request that the tables be checked (with the IMMEDIATE CHECKED clause of the SET CONSTRAINTS statement). Inconsistent data will be indicated by an arithmetic error, which prevents the constraint from being evaluated.

Recommendation: Use the default setting of no, unless you specifically require queries to be processed that include arithmetic exceptions. Then specify the value of yes. This situation can occur if you are processing SQL statements that, on other database managers, provide results regardless of the arithmetic exceptions that occur.

Default Degree (dft_degree)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	1 [-1, 1 – 32 767]
Related Parameters	“Maximum Query Degree of Parallelism (max_querydegree)” on page 559

This parameter specifies the default value for the CURRENT DEGREE special register and the DEGREE bind option.

The default value is 1.

A value of 1 means no intra-partition parallelism. A value of -1 means the optimizer determines the degree of intra-partition parallelism based on the number of processors and the type of query.

The degree of intra-partition parallelism for a SQL statement is specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option. The maximum runtime degree of intra-partition parallelism for an active application is specified using the SET RUNTIME DEGREE command. The Maximum Query Degree of Parallelism (max_querydegree) configuration parameter specifies the maximum query degree of intra-partition parallelism for all SQL queries.

The actual runtime degree used is the lowest of:

- *max_querydegree* configuration parameter
- application runtime degree
- SQL statement compilation degree

Default Query Optimization Class (dft_queryopt)

Configuration Type Database

Parameter Type Configurable

Default [Range] 5 [0 – 9]

Unit of Measurement Query Optimization Class (see below)

The query optimization class is used to direct the optimizer to use different degrees of optimization when compiling SQL queries. This parameter provides additional flexibility by setting the default query optimization class used when neither the SET CURRENT QUERY OPTIMIZATION statement nor the QUERYOPT bind command are used.

The query optimization classes currently defined are:

0 - minimal query optimization.
1 - roughly comparable to DB2 Version 1.
2 - slight optimization. Specifies a level of optimization higher than that of Version 1, but at significantly less optimization cost than levels 3 and above, especially for very complex queries.
3 - moderate query optimization.
5 - significant query optimization with heuristics to limit the effort expended on selecting an access plan. This is the default.
7 - significant query optimization.
9 - maximal query optimization

Recommendation: For more information and guidance for selecting a suitable query optimization class, see “Adjusting the Optimization Class” on page 283.

For more information on how a program can retrieve and modify database configuration parameters, see *API Reference*.

Number of Frequent Values Retained (num_freqvalues)

Configuration Type Database

Parameter Type Configurable

Default [Range] 10 [0 – 32 767]

Unit of Measure Counter

Related Parameters

- "Number of Quantiles for Columns (num_quantiles)"
- "Statistics Heap Size (stat_heap_sz)" on page 485

This parameter allows you to specify the number of "most frequent values" that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command. Increasing the value of this parameter increases the amount of statistics heap (*stat_heap_sz*) used when collecting statistics.

The "most frequent value" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the SQL optimizer but requires additional catalog space. When 0 is specified, no frequent-value statistics are retained, even if you request that distribution statistics be collected.

Updating this parameter can help the optimizer obtain better selectivity estimates for some predicates (=, <, >, IS NULL, IS NOT NULL) over data that is non-uniformly distributed. More accurate selectivity calculations may result in the choice of more efficient access plans.

After changing the value of this parameter, you need to:

- Run the RUNSTATS command after all users have disconnected from the database and you have reconnected to the database
- Rebind any packages containing static SQL.

For more information, see "Collecting and Using Distribution Statistics" on page 318.

Recommendation: In order to update this parameter you should determine the degree of non-uniformity in the most important columns (in the most important tables) that typically have selection predicates. This can be done using an SQL SELECT statement that provides an ordered ranking of the number of occurrences of each value in a column. You should not consider uniformly distributed, unique, long, or LOB columns. A reasonable practical value for this parameter lies in the range of 10 to 100.

Note that the process of collecting frequent value statistics requires significant CPU and memory (*stat_heap_sz*) resources.

Number of Quantiles for Columns (num_quantiles)

Configuration Type Database

Parameter Type Configurable

Default [Range] 20 [0 – 32 767]

Unit of Measure Counter

Related Parameters

- "Number of Frequent Values Retained (num_freqvalues)" on page 544
- "Statistics Heap Size (stat_heap_sz)" on page 485

This parameter controls the number of quantiles that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command. Increasing the value of this parameter increases the amount of statistics heap (*stat_heap_sz*) used when collecting statistics.

The "quantile" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the SQL optimizer but requires additional catalog space. When 0 or 1 is specified, no quantile statistics are retained, even if you request that distribution statistics be collected.

Updating this parameter can help obtain better selectivity estimates for range predicates over data that is non-uniformly distributed. Among other optimizer decisions, this information has a strong influence on whether an index scan or a table scan will be chosen. (It is more efficient to use a table scan to access a range of values that occur frequently and it is more efficient to use an index scan for a range of values that occur infrequently.)

After changing the value of this parameter, you need to:

- Run the RUNSTATS command after all users have disconnected from the database and you have reconnected to the database
- Rebind any packages containing static SQL.

For more information, see "Collecting and Using Distribution Statistics" on page 318.

Recommendation: This default value for this parameter guarantees a maximum estimation error of approximately 2.5% for any single-sided range predicate (>, >=, <, or <=), and a maximum error of 5% for any BETWEEN predicate. A rough rule of thumb for determining the number of quantiles is:

- Determine the maximum error that is tolerable in estimating the number of rows of any range query, as a percentage, P
- The number of quantiles should be approximately 100/P if most of your predicates are BETWEEN predicates, and 50/P if most of your predicates are other types of range predicates (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and of 2% for ">" predicates. A reasonable practical value for this parameter lies in the range of 10 to 50.

Communications

The following groups of parameters provide information about using DB2 in a client/server environment:

- “Communication Protocol Setup”
- “Distributed Services” on page 549.
- “DB2 Discovery” on page 553

Communication Protocol Setup

You can use the following parameters to configure your database clients and database servers:

- “NetBIOS Workstation Name (nname)”
- “TCP/IP Service Name (svcename)” on page 547
- “APPC Transaction Program Name (tpname)” on page 547
- “IPX/SPX File Server Name (fileserver)” on page 548
- “IPX/SPX DB2 Server Object Name (objectname)” on page 548
- “IPX/SPX Socket Number (ipx_socket)” on page 549

NetBIOS Workstation Name (nname)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

This parameter allows you to assign a name for each node, or workstation, in the NetBIOS LAN environment. This *nname* is the basis for the actual NetBIOS names that will be registered for a NetBIOS protocol workstation. You must ensure that each node in the NetBIOS LAN environment has a unique nname.

Since the NetBIOS protocol establishes its communication connections using these NetBIOS names, the *nname* parameter must be set for both client and server nodes.

Client applications must know the *nname* of the server that contains the database to be accessed. The server's *nname* must be cataloged into the client node directories as the “server NNAME” parameter using the CATALOG NETBIOS NODE command, for example.

If *nname* at the server node changes to a new name, all clients accessing databases on that server must catalog this new name for the server.

TCP/IP Service Name (svcename)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Partitioned Database Server with local and remote clients

Database Server with local and remote clients

Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

This parameter contains the name of the TCP/IP port which a database server will use to await communications from remote client nodes. This name must be the first of two consecutive ports reserved for use by the database manager; the second port is used to handle interrupt requests from down-level clients.

In order to accept connection requests from a database client using TCP/IP, the database server must be listening on a port designated to that server. The system administrator for the database server must reserve a port (number *n*) and define its associated TCP/IP service name in the services file at the server. If the database server needs to support requests from down-level clients, a second port (number *n+1*, for interrupt requests) needs to be defined in the services file at the server.

The database server port (number *n*) and its TCP/IP service name need to be defined in the services file on the database client. Down-level clients also require the interrupt port (number *n+1*) to be defined in the client's services file.

The location of the services file depends on your operating environment. For example:

- In UNIX — /etc/services
- In OS/2 — \tcpip\etc\services.

The *svcename* parameter should be set to the service name associated with the main connection port so that when the database server is started, it can determine on which port to listen for incoming connection requests. If you are supporting or using a down-level client, the service name for the interrupt port is not saved in the configuration file. The interrupt port number can be derived based on the main connection port number (*interrupt port number* = *main connection port* + 1).

See the *Quick Beginnings* for more information about setting up TCP/IP for database servers.

APPC Transaction Program Name (tpname)

Configuration Type Database manager

Applies to Database Server with local and remote clients

Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

This parameter defines the name of the remote transaction program that the database client must use when it issues an allocate request to the database server when using the APPC communication protocol. This parameter must be set in the configuration file at the database server.

This parameter must be the same as the transaction program name that is configured in the SNA transaction program definition. See the *Quick Beginnings* for more information about setting up APPC for your DB2 product.

Recommendation: The only accepted characters for use in this name are:

- Alphabetics (A through Z; or a through z)
- Numerics (0 through 9)
- Dollar sign ($), number sign (#), at sign (@), and period (.).

IPX/SPX File Server Name (fileserver)

Configuration Type Database manager

Applies to Database Server with local and remote clients

Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

Related Parameters

- "IPX/SPX DB2 Server Object Name (objectname)"
- "IPX/SPX Socket Number (ipx_socket)" on page 549

This parameter specifies the name of the NetWare** fileserver where the internetwork address of the database manager is registered. The internetwork address of the database manager is stored in the bindery at the NetWare file server. If the registered fileserver name changes, all clients that access the server instance must:

- UNCATALOG the server node
- CATALOG the server node, specifying the new fileserver name.

For more information, see the *Quick Beginnings* manual appropriate for your platform.

IPX/SPX DB2 Server Object Name (objectname)

Configuration Type Database manager

Applies to Database Server with local and remote clients

Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

Related Parameters

- "IPX/SPX File Server Name (fileserver)"
- "IPX/SPX Socket Number (ipx_socket)" on page 549

This parameter provides the name of the database manager instance in an IPX/SPX network. Each server instance registered to a NetWare fileserver must have a unique name. If this name changes at the database server, all clients that access the server must uncatalog the server node and recatalog it again, specifying the new object name.

IPX/SPX Socket Number (ipx_socket)

Configuration Type Database manager

Applies to Database Server with local and remote clients

Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 879E [879E – 87A2] To ensure that there are no conflicts, five socket numbers (879E to 87A2) are uniquely registered with Novell for use by DB2.

Related Parameters

- "IPX/SPX File Server Name (fileserver)" on page 548
- "IPX/SPX DB2 Server Object Name (objectname)" on page 548

This parameter specifies a "well-known" socket number and represents the connection end point in a DB2 server's internetwork address. The socket number must be unique for each DB2 server instance on a given machine, and unique among all Novell** IPX/SPX applications running on this same machine. This is to guarantee that the DB2 server is able to listen to incoming IPX/SPX connections using this socket number.

Distributed Services

You can use the following parameters to configure your database clients and database servers to make use of DCE Directory services:

- "Directory Services Type (dir_type)" on page 550
- "Directory Path Name in DCE Namespace (dir_path_name)" on page 550
- "Object Name in DCE Namespace (dir_obj_name)" on page 551
- "Routing Information Object Name (route_obj_name)" on page 552
- "Default Client Communication Protocol (dft_client_comm)" on page 552
- "Default Client Adapter Number (dft_client_adpt)" on page 553

For information about how DB2 uses DCE directories, see Appendix F, "Using Distributed Computing Environment (DCE) Directory Services" on page 647.

Directory Services Type (dir_type)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- UNIX and OS/2 Client
- UNIX and OS/2 Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] NONE [NONE; DCE]

Related Parameters

- "Object Name in DCE Namespace (dir_obj_name)" on page 551
- "Directory Path Name in DCE Namespace (dir_path_name)"
- "Routing Information Object Name (route_obj_name)" on page 552
- "Default Client Communication Protocol (dft_client_comm)" on page 552
- "Default Client Adapter Number (dft_client_adpt)" on page 553

This parameter indicates whether or not DCE directory services is used.

If this parameter is set to **NONE**, only local directory files will be searched for the target of the CONNECT or ATTACH requests. However, you can still use the *dir_path_name* and *dir_obj_name* parameters to record the name of your database instance and databases in the DCE namespace.

If this parameter is set to **DCE**, then when an application running within this database manager instance cannot find the target of its CONNECT or ATTACH requests, the DCE directory will be searched.

The API constants for this parameter are:

- SQLF_DIRTYPE_NONE
- SQLF_DIRTYPE_DCE

Directory Path Name in DCE Namespace (dir_path_name)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- UNIX and OS/2 Client
- UNIX and OS/2 Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default /.:/subsys/database/

Related Parameters

- "Object Name in DCE Namespace (dir_obj_name)" on page 551
- "Directory Services Type (dir_type)" on page 550
- "Routing Information Object Name (route_obj_name)" on page 552

The unique name of the database manager instance in the global namespace is made up of this value and the value in the *dir_obj_name* parameter.

All client applications running within this instance also use it as the default path name for their CONNECT or ATTACH requests, unless it is overridden by the value of the *DB2DIRPATHNAME* environment variable.

Recommendation: Use the name provided by your DCE administrator.

Object Name in DCE Namespace (dir_obj_name)

Configuration Type Database manager, Database

Applies to

- Database Server with local and remote clients
- UNIX and OS/2 Client
- UNIX and OS/2 Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

Related Parameters

- "Directory Services Type (dir_type)" on page 550
- "Directory Path Name in DCE Namespace (dir_path_name)" on page 550

The object name representing your database manager instance (or your database) in the directory. The concatenation of this value and the *dir_path_name* value yields a global name that uniquely identifies the database manager instance or database in the namespace governed by the directory services specified in the *dir_type* parameter.

This parameter is only meaningful if the *dir_path_name* parameter is specified.

The total length of the configuration parameters *dir_path_name* and *dir_obj_name* must be less than 255 characters.

Recommendation: For more information, see Appendix F, "Using Distributed Computing Environment (DCE) Directory Services" on page 647.

Routing Information Object Name (route_obj_name)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

Related Parameters

- "Directory Path Name in DCE Namespace (dir_path_name)" on page 550
- "Directory Services Type (dir_type)" on page 550

This parameter specifies the name of the default routing information object entry that will be used by all client applications attempting to access a DRDA server. It applies to OS/2 and UNIX-based environments only.

If the value of this parameter starts with */.:/* or */.../*, then the value will be used as is. Otherwise, it will be appended to the *dir_path_name* parameter (or **DB2DIRPATHNAME** environment variable) value to form the full name of the routing information object.

You can use the environment variable DB2ROUTE to override this default.

This parameter is only meaningful if the *dir_type* parameter is set to DCE.

Recommendation: For more information, see Appendix F, "Using Distributed Computing Environment (DCE) Directory Services" on page 647.

Default Client Communication Protocol (dft_client_comm)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] Null [Null; TCPIP; APPC; IPXSPX (OS/2 only); NETBIOS (OS/2 only)]

Related Parameters "Directory Services Type (dir_type)" on page 550

This parameter indicates the communication protocols that the client applications on this instance can use for remote connections. Its content is a character string, made up of one or more tokens. If you are specifying more than one token, separate them with a comma. The order of the tokens is significant in terms of preference.

This parameter can only be used with DCE, and applies to OS/2 and UNIX-based environments only.

You can temporarily override the value of this parameter by setting the DB2CLIENTCOMM environment variable.

If the value of this parameter is NULL and the environment variable has not been set, the first protocol specified in the server's global directory object is used.

This parameter is ignored if *dir_type* is set to NONE.

Recommendation: The protocol that is used most often should be specified first.

Default Client Adapter Number (dft_client_adpt)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Client
- Database Server with local clients

Parameter Type Configurable

Default [Range] 0 [0–15]

Related Parameters

- "Default Client Communication Protocol (dft_client_comm)" on page 552.
- "Directory Services Type (dir_type)" on page 550. (When dir_type is set to DCE.)

This parameter defines the default client adapter number for the NETBIOS protocol whose server nname is extracted from DCE Cell Directory Services (CDS). This parameter is applicable to the OS/2 environment only.

This parameter can only be used with DCE.

You can temporarily override the value of this parameter by setting the DB2CLIENTADPT environment variable. If this environment variable contains a non-numeric or out-of-range number, adapter number 0 (zero) is used.

DB2 Discovery

You can use the following parameters to establish DB2 Discovery:

- "Discover Database (discover_db)" on page 554
- "Discovery Mode (discover)" on page 554
- "Search Discovery Communications Protocols (discover_comm)" on page 555
- "Discover Server Instance (discover_inst)" on page 555

Discover Database (discover_db)

Configuration Type Database

Parameter Type Configurable

Default [Range] Enable [Disable, Enable]

This parameter is used to prevent information about a database from being returned to a client when a discovery request is received at the server.

The default for this parameter is that discovery is enabled for this database.

By changing this parameter value to disable, it is possible to hide databases with sensitive data from the discovery process. This can be done in addition to other database security controls on the database.

Discovery Mode (discover)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] search [disable, known, search]

Related Parameters “Search Discovery Communications Protocols (discover_comm)” on page 555

This parameter defines the default discovery action when DB2 starts.

The default discovery action is “search.” When this value is specified, DB2 Discovery uses the protocols specified by the *discover_comm* parameter to search the network for databases.

If the value “known” is specified, the Client Configuration Assistant allows you to specify the connection information for a DB2 server on the network, and returns the databases that it finds on that server.

By selecting “disable” for this parameter, DB2 Discovery is not started on administration servers, and requests to administration servers and non-administration servers are not honored.

For more information on DB2 Discovery, see the *Quick Beginnings* manual appropriate to your platform.

Search Discovery Communications Protocols (discover_comm)

Configuration Type Database manager

Applies To

- Client
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] None [Any combination of IPXSPX, NETBIOS, NPIPE, TCPIP]

Related Parameters “Discovery Mode (discover)” on page 554

This parameter defines the communications protocols that clients use to issue search discovery requests, and servers use to listen for search discovery requests. More than one protocol may be specified, separated by commas; or, the parameter may be left blank.

The default for this parameter is "None" meaning that there are no communications protocols.

Discover Server Instance (discover_inst)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] enable [enable, disable]

This parameter specifies whether this instance can be detected by DB2 Discovery. The default, “enable,” specifies that the instance can be detected, while “disable” prevents the instance from being discovered.

For more information on DB2 Discovery, see the *Quick Beginnings* manual appropriate to your platform.

Parallel

The following parameters relate to parallel operations and partitioned database environments:

- “Connection Elapse Time (conn_elapse)” on page 556
- “Number of FCM Message Anchors (fcm_num_anchors)” on page 556
- “Number of FCM Buffers (fcm_num_buffers)” on page 557
- “Number of FCM Connection Entries (fcm_num_connect)” on page 558
- “Number of FCM Request Blocks (fcm_num_rqb)” on page 558
- “Node Connection Retries (max_connretries)” on page 559

- “Maximum Query Degree of Parallelism (max_querydegree)” on page 559
- “Maximum Time Difference Among Nodes (max_time_diff)” on page 560
- “Enable Intra-Partition Parallelism (intra_parallel)” on page 560
- “Start and Stop Timeout (start_stop_time)” on page 561

Connection Elapse Time (conn_elapse)

Configuration Type Database manager

Applies To Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 10 [0–100]

Unit of Measure Seconds

Related Parameters “Node Connection Retries (max_connretries)” on page 559

This parameter specifies the number of seconds within which a TCP/IP connection is to be established between two database partition servers. If the attempt completes within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the *max_connretries* parameter and always times out, an error is issued.

Number of FCM Message Anchors (fcm_num_anchors)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 [-1, 128–*fcm_num_rqb*]

On non-partitioned database systems, *parallel_enable* parameter must be active before this parameter can be used.

Related Parameters

- “Number of FCM Request Blocks (fcm_num_rqb)” on page 558
- “Enable Intra-Partition Parallelism (intra_parallel)” on page 560

This parameter specifies the number of FCM *message anchors*. Agents use the message anchors to send messages among themselves. The default (-1) indicates 75 percent of the value specified for *fcm_num_rqb*.

Number of FCM Buffers (fcm_num_buffers)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 1 024 or 4 096 [128–65 300]

- Database Server with local and remote clients: the default is 4 096
- Database Server with local clients: the default is 1 024
- Partitioned Database Server with local and remote clients: the default is 4 096

On non-partitioned database systems, the *parallel_enable* parameter must be active before this parameter can be used.

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within the database servers in a partitioned database environment. For more information about FCM, see "Enable FCM Communications" on page 70.

If you have multiple logical nodes on a processor, you may find it necessary to increase the value of this parameter. You may also find it necessary to increase the value of this parameter if you run out of message buffers because of the number of users on the system, the number of nodes on the system, or the complexity of the applications.

If you are using multiple logical nodes, on non-AIX systems, one pool of *fcm_num_buffers* buffers is shared by all the multiple logical nodes on the same machine, while on AIX:

- If there is enough room in the general memory that is used by the database manager, the FCM buffer heap will be allocated from there. In this situation, each node will have *fcm_num_buffers* buffers of its own; the nodes will not share a pool of FCM buffers (this is new to DB2 Version 5).
- If there is not enough room in the general memory that is used by the database manager, the FCM buffer heap will be allocated from a separate memory area (AIX shared memory set), that is shared by all the multiple logical nodes on the same machine. One pool of *fcm_num_buffers* will be shared by all the multiple logical nodes on the same machine. This is the same as non-AIX systems and is also the same as DB2 Parallel Edition Version 1.2 on AIX.

Recommendation for existing Parallel Edition customers on AIX: If you are using multiple logical nodes, the value of *fcm_num_buffers* you used in Parallel Edition Version 1.2 may now result in significantly more storage being used per machine. For

example, a four-node multiple logical node configuration may end up with four times as many FCM buffers as before.

Re-examine the value you are using; consider how many FCM buffers in total will be allocated on the machine (or machines) where the multiple logical nodes reside. You may want to change *fcm_num_buffers* to account for the behaviour described above.

Number of FCM Connection Entries (fcm_num_connect)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 [-1, 128–*fcm_num_rqb*]

On non-partitioned database systems, the *parallel_enable* parameter must be active before this parameter can be used.

Related Parameters "Number of FCM Request Blocks (fcm_num_rqb)"

This parameter specifies the number of FCM *connection entries*. Agents use connection entries to pass data among themselves. The default (-1) indicates 75 percent of the value specified for *fcm_num_rqb*.

Number of FCM Request Blocks (fcm_num_rqb)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 1024 or 2048 [128–120000]

- Database Server with local and remote clients: the default is 2048
- Database Server with local clients: the default is 1024
- Partitioned Database Server with local and remote clients: the default is 2048

On non-partitioned database systems, the *parallel_enable* parameter must be active before this parameter can be used.

This parameter specifies the number of FCM *request blocks*. Request blocks are the media through which information is passed between the FCM daemon and an agent, or between agents.

The requirement for request blocks will vary according to the number of users on the system, the number of database partition servers in the system, and the complexity of queries that are run. Initially, start with the default number, and use the results from the Database System Monitor when fine tuning this parameter.

Node Connection Retries (max_connretries)

Configuration Type Database manager

Applies To Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 5 [0–100]

Related Parameters "Connection Elapse Time (conn_elapse)" on page 556

If the attempt to establish communication between two database partition servers fails (for example, the value specified by the *conn_elapse* parameter is reached), *max_connretries* specifies the number of connection retries that can be made to a database partition server. If the value specified for this parameter is exceeded, an error is returned.

Maximum Query Degree of Parallelism (max_querydegree)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (ANY) [ANY, 1–32 767] (ANY means system determined)

Related Parameters

- "Default Degree (dft_degree)" on page 542
- "Enable Intra-Partition Parallelism (intra_parallel)" on page 560

This parameter specifies the maximum degree of intra-partition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a partition when the statement is executed. For a partitioned database system, this parameter applies to the degree of database partition parallelism. The *parallel_enable* parameter must be set to "YES" to enable the database partition to use intra-partition parallelism.

The default value for this configuration parameter is -1. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

Note: The degree of parallelism for a SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option.

The maximum query degree of parallelism for an active application is specified using the SET RUNTIME DEGREE command. The actual runtime degree used is the lower of:

- *max_querydegree* configuration parameter
- Application runtime degree
- SQL statement compilation degree

An exception regarding the determination of the actual query degree of parallelism occurs when creating an index. In this case, if *parallel_enable* is "YES" and the table is large enough to benefit from the use of multiple processors, then creating an index uses the number of online processors plus one. There is no effect from the other parameter, bind option, or special register mentioned above.

Maximum Time Difference Among Nodes (max_time_diff)

Configuration Type	Database manager
Applies To	Partitioned Database Server with local and remote clients
Parameter Type	Configurable
Default [Range]	60 [1–1 440]
Unit of Measure	Minutes

Each database partition server has its own system clock. This parameter specifies the maximum time difference, in minutes, that is permitted among the database partition servers listed in the `db2nodes.cfg` file.

If two or more database partition servers are associated with a transaction and their clocks are not synchronized to within the time specified by this parameter, the transaction is rejected and a warning or an error message is logged in the `db2diag.log` file. (The transaction is rejected only if data modification is associated with it.)

DB2 Universal Database Extended Enterprise Edition uses *Coordinated Universal Time*, (UTC) so different time zones are not a consideration when you set this parameter. The Coordinated Universal Time is the same as Greenwich Mean Time.

Enable Intra-Partition Parallelism (intra_parallel)

Configuration Type	Database manager

Applies To

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] SYSTEM (-1) [SYSTEM (-1), NO (0), YES (1)]

A value of -1 causes the parameter value to be set to “YES” or “NO” based on the hardware on which the database manager is running.

Related Parameters “Maximum Query Degree of Parallelism (max_querydegree)” on page 559

This parameter specifies whether the database manager can use intra-partition parallelism.

In a symmetric multiprocessor (SMP) environment, the default for this parameter is "YES". In a non-SMP environment, the default for this parameter is "NO". This parameter can be used on both partitioned and non-partitioned database systems.

Some of the operations that can take advantage of parallel performance improvements when this parameter is "YES" include database queries and index creation.

Note: If you change this parameter value, packages may be rebound to the database. If this occurs, a performance degradation may occur.

Start and Stop Timeout (start_stop_time)

Configuration Type Database manager

Applies To Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 10 [1–1 440]

Unit of Measure Minutes

This parameter is applicable in a partitioned database environment only. It specifies the time, in minutes, within which all database partition servers must respond to a DB2START or a DB2STOP command. It is also used as the timeout value during an ADDNODE operation.

Database partition servers that do not respond to a DB2START command within the specified time send a message to the `db2start` error log in the `log` subdirectory of the `sqllib` subdirectory of the `$HOME` directory. You should issue a DB2STOP on these nodes before restarting them.

Database partition servers that do not respond to a DB2STOP command within the specified time send a message to the `db2stop` error log in the `log` subdirectory of the

sqllib subdirectory of the $HOME directory. You can either issue DB2STOP for each database partition server that does not respond, or for all of them. (Those that are already stopped will return stating that they are stopped.)

Instance Management

A number of parameters can help you manage your database manager instances. These are grouped into the following categories:

- "Diagnostic"
- "Database System Monitor Parameters" on page 563
- "System Management" on page 564
- "Instance Administration" on page 570

Diagnostic

The following parameters allow you to control diagnostic information available from the database manager:

- "Diagnostic Error Capture Level (diaglevel)"
- "Diagnostic Data Directory Path (diagpath)" on page 563

Diagnostic Error Capture Level (diaglevel)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] 3 [0 – 4]

Related Parameters "Diagnostic Data Directory Path (diagpath)" on page 563

The type of diagnostic errors recorded in the error log file is determined by this parameter. Valid values are:

0 – No diagnostic data captured

1 – Severe errors only

2 – All errors

3 – All errors and warnings

4 – All errors, warnings and informational messages

It is the *diagpath* configuration parameter that is used to specify the directory that will contain the error log file, event log (on Windows NT only), alert log file, and any dump files that may be generated based on the value of the *diaglevel* parameter.

Recommendation: You may wish to increase the value of this parameter to gather additional problem determination data to help resolve a problem.

Diagnostic Data Directory Path (diagpath)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid path name]

Related Parameters "Diagnostic Error Capture Level (diaglevel)" on page 562

This parameter allows you to specify the fully qualified path for DB2 diagnostic information. This directory coulld possibly contain dump files, trap files, an error log and an alert log file, depending on your platform.

If this parameter is null, the diagnostic information will be written to files in one of the following directories or folders:

- For OS/2, Windows, Windows 95, and Windows NT:
  - If the DB2INSTPROF environment variable or keyword is **not** set, information will be written to `x:\SQLLIB\DB2INSTANCE`, where `x:` is the drive reference in the DB2PATH environment variable or keyword and `DB2INSTANCE` is the name of the instance owner.

    Note: The directory does not have to be named SQLLIB.
  - If the DB2INSTPROF environment variable or keyword is set, information will be written to `x:\DB2INSTPROF\DB2INSTANCE`, where DB2INSTPROF is the name of the instance profile directory.
- For UNIX-based environments: `INSTHOME/sqllib/db2dump`, where `INSTHOME` is the home directory of the instance owner.
- For Macintosh environments: DB2 folder.

Recommendation: Use the default or have a centralized location for the diagpath of multiple instances.

In a multinode environment, the path you specify must reside on a shared file system.

Database System Monitor Parameters

The following parameter allows you to control various aspects of the database system monitor:

- "Default Database System Monitor Switches (dft_monswitches)" on page 564

Default Database System Monitor Switches (dft_monswitches)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default All switches turned off

This parameter is unique in that it allows you to set a number of switches which are each internally represented by a bit of the parameter. Depending on the interface you are using to update the database manager configuration, you may be able to update this parameter directly. You may also update each of these switches independently by setting the following parameters:

dft_mon_uow Default value of the snapshot monitor's unit of work (UOW) switch

dft_mon_stmt Default value of the snapshot monitor's statement switch

dft_mon_table Default value of the snapshot monitor's table switch

dft_mon_bufpool Default value of the snapshot monitor's buffer pool switch

dft_mon_lock Default value of the snapshot monitor's lock switch

dft_mon_sort Default value of the snapshot monitor's sort switch

For more information about the snapshot monitor and how it uses monitor switches, see the *System Monitor Guide and Reference*.

Recommendation: Any switch that is turned ON instructs the database manager to collect monitor data related to that switch. Collecting additional monitor data increases database manager overhead which can impact system performance.

All monitoring applications inherit these default switch settings when the application issues its first monitoring request (for example, setting a switch, activating the event monitor, taking a snapshot). You should turn on a switch in the configuration file, only if you want to collect data starting from the moment the database manager is started. (Otherwise, each monitoring application can set its own switches and the data it collects becomes relative to the time its switches are set.)

System Management

The following parameters relate to system management:

- "Communications Bandwidth (comm_bandwidth)" on page 565
- "CPU Speed (cpuspeed)" on page 565
- "Maximum Number of Concurrently Active Databases (numdb)" on page 566
- "Transaction Processor Monitor Name (tp_mon_name)" on page 567
- "Machine Node Type (nodetype)" on page 568

- “Default Charge-Back Account (dft_account_str)” on page 568
- “Java Development Kit 1.1 Installation Path (jdk11_path)” on page 569

Communications Bandwidth (comm_bandwidth)

Configuration Type Database manager

Applies to

- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 [.1 – 100 000]

A value of -1 causes the parameter value to be reset to the default. The default value is calculated based on whether a high speed switch is being used. If the switch is used, -1 evaluates to 100MB per second. If the switch is not used, -1 evaluates to 2MB per second.

Unit of Measure Megabytes per second

The value calculated for the communications bandwidth, in megabytes per second, is used by the SQL optimizer to estimate the cost of performing certain operations between the nodes of an MPP system. The optimizer does not model the cost of communications between a client and a server, so this parameter should reflect only the nominal bandwidth between MPP nodes, if any.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The communications bandwidth is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

CPU Speed (cpuspeed)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] -1 [1e-10 – 1] A value of -1 will cause the parameter value to be reset based on the running of the measurement program.

The CPU speed, in milliseconds per instruction, is used by the SQL optimizer to estimate the cost of performing certain operations. The value of this parameter is set automatically when you install the database manager based on one of the following:

- Data from the `db2spec.dat` file located in the `cfg` sub-directory. For example, in AIX-based environments this file is located in the `$DB2INSTANCE/sqllib/cfg` directory. This file contains SPECint92** benchmark results. Information from this file will be used if SPECint92 benchmark data can be located for both of the following:
  - The machine on which the database manager instance is running
  - The IBM RISC System/6000 model 530H. (SPECint92 data is used to calibrate machines relative to the 530H which is why the data for the 530H is required.)

  Note: You may update the `db2spec.dat` file if you have SYSADM authority. You should carefully follow the instructions contained in that file.
- Output from a program designed to measure CPU speed. This program is executed, if benchmark results are not available for any of the following reasons:
  - The platform does not have support for the `db2spec.dat` file
  - The `db2spec.dat` file is **not** found
  - The data for the IBM RISC System/6000 model 530H is not found in the file
  - The data for your machine is not found in the file.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware. By setting it to -1, *cpuspeed* will be re-computed.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The CPU speed is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Maximum Number of Concurrently Active Databases (numdb)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX	8 [1 – 256]
OS/2 and NT Database Server with local and remote clients	8 [1 – 256]

OS/2 and NT Database Server with local clients
3 [1 – 256]

Unit of Measure Counter

This parameter specifies the number of local databases that can be concurrently active (that is, have applications connected to them). In a partitioned database environment, it limits the number of active database partitions on a database partition server, whether that server is the coordinator node for the application or not.

Since each database takes up storage and an active database uses a new shared memory segment, you can reduce system resource usage by limiting the number of separate databases on your machine. However, arbitrarily reducing the number of databases is not the answer. That is, putting all data, no matter how unrelated, in one database will reduce disk space, but may not be a good idea. It is generally a good practice to only keep functionally related information in the same database.

Recommendation: It is generally best to set this value to the actual number of databases that are already defined to the database manager and to add a reasonable increment to account for future growth in the number of databases over the short term (for example, 6 months to 1 year). The actual increment should not be excessively large, but it should allow you to add new databases without having to frequently update this parameter.

Changing the *numdb* parameter may impact the total amount of memory allocated. As a result, frequent updates to this parameter are not recommended. When updating this parameter, you should consider the other configuration parameters that can allocate memory for a database or an application connected to that database, including:

- “Buffer Pool Size (buffpage)” on page 470
- “Maximum Storage for Lock List (locklist)” on page 477
- “Application Heap Size (applheapsz)” on page 484
- “Application Control Heap Size (app_ctl_heap_sz)” on page 480
- “Sort Heap Size (sortheap)” on page 482
- “Statement Heap Size (stmtheap)” on page 484
- “Application Support Layer Heap Size (aslheapsz)” on page 492
- “Database Heap (dbheap)” on page 472
- “Database System Monitor Heap Size (mon_heap_sz)” on page 495
- “Statistics Heap Size (stat_heap_sz)” on page 485

Transaction Processor Monitor Name (tp_mon_name)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Valid Values CICS; ENCINA; blank or some other value (for UNIX, OS/2, and Windows NT; none for Solaris or SINIX)

This parameter identifies the name of the transaction processing (TP) monitor product being used. If applications are run in a CICS environment, this parameter should be set to "CICS"; if Encina Monitor is being used, this parameter should be set to "ENCINA."

In OS/2 and NT environments, this parameter contains the path and name of the DLL in an external transaction manager product containing functions `ax_reg` and `ax_unreg`, if an XA Distributed Transaction Processing environment is being used. Specify "*dll-name*:C" for CICS, or "*dll-name*:E" for ENCINA. The maximum length of the string that can be specified for this parameter is 19 characters.

Machine Node Type (nodetype)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Informational

This parameter provides information about the DB2 products which you have installed on your machine and, as a result, information about the type of database manager configuration. The following are the possible values returned by this parameter and the products associated with that node type:

- **Database Server with local and remote clients** – a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers. (The API constant for this *nodetype* is SQLF_NT_SERVER.)
- **Client** – a database client capable of accessing remote database servers (The API constant for this *nodetype* is SQLF_NT_REQUESTER.)
- **Database Server with local clients** – a DB2 relational database management system, supporting local database clients and capable of accessing other, remote database servers. (The API constant for this *nodetype* is SQLF_NT_STAND_REQ.)
- **Partitioned Database Server with local and remote clients** – a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers, and capable of partition parallelism. (The API constant for this *nodetype* is SQLF_NT_MPP.)

Default Charge-Back Account (dft_account_str)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients

- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

With each application connect request, an accounting identifier consisting of a DB2 Connect-generated prefix and the user supplied suffix is sent from the application requester to a DRDA application server. This accounting information provides a mechanism for system administrators to associate resource usage with each user access.

The suffix is supplied by the application program calling the `sqlesact()` API or the user setting the environment variable DB2ACCOUNT. If a suffix is not supplied by either the API or environment variable, DB2 Connect uses the value of this parameter as the default suffix value. This parameter is particularly useful for down-level database clients (anything prior to version 2) that do not have the capability to forward an accounting string to DB2 Connect.

Recommendation: Set this accounting string using the following:

- Alphabetics (A through Z)
- Numerics (0 through 9)
- Underscore (_).

Java Development Kit 1.1 Installation Path (jdk11_path)

Configuration Type Database manager

Applies To

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] Null [Valid path]

Related Parameters

- “Maximum Java Interpreter Heap Size (java_heap_sz)” on page 498

This parameter specifies the directory under which the Java Development Kit 1.1 is installed. The CLASSPATH and other environment variables used by the Java interpreter are computed from the value of this parameter.

Because there is no default for this parameter, you should specify a value for this parameter when you install the Java Development Kit.

Instance Administration

The following parameters relate to security and administration of your database manager instance:

System Administration Authority Group Name (sysadm_group)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default

UNIX	Null
OS/2	Null
Windows 95	Null
Windows NT	Null

Related Parameters

- "System Control Authority Group Name (sysctrl_group)" on page 571
- "System Maintenance Authority Group Name (sysmaint_group)" on page 572

System administration (SYSADM) authority is the highest level of authority within the database manager and controls all database objects. This parameter defines the group name with SYSADM authority for the database manager instance.

SYSADM authority is determined by the security facilities used in a specific operating environment. The following apply when system security (that is, authorization is CLIENT, SERVER, or DCS. Considerations for DCE security are described below.)

- In Windows 95 the SYSADM group must be NULL.

  This parameter must be "NULL" for Windows 95 clients when system security is used because Windows 95 does not store group information, thereby providing no way of determining if a user is a member of a designated SYSADM group. When a

group name is specified, no user is considered to be a member of it and no user is considered to have administration authority.

- For Windows NT, this parameter can be set to any local group that has a name of 16 characters or fewer, and is defined in the Windows NT security database. If "NULL" is specified for this parameter, all members of the Administrators group have SYSADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the SYSADM group defaults to the primary group of the instance owner.

  If the value is not "NULL," the SYSADM group can be any valid UNIX group name.
- In OS/2, if the value specified for this parameter is "Null," users defined as administrators in user profile management have SYSADM authority.

  If a group name is specified for this parameter, only users who belong to the group have SYSADM authority. The group specified can be any of the User Profile Management (user profile management) groups;. For more information on User Profile Management groups, see your *Quick Beginnings for OS/2*. .

If DCE security is used and *sysadm_group* is "NULL," the default DCE group name DB2ADMIN is used. A valid DCE principal whose authid mapping is DB2ADMIN must already exist. You can specify a different group name (this also applies for Windows 95 clients).

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSADM_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

System Control Authority Group Name (sysctrl_group)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

Related Parameters

- "System Administration Authority Group Name (sysadm_group)" on page 570
- "System Maintenance Authority Group Name (sysmaint_group)" on page 572

This parameter defines the group name with system control (SYSCTRL) authority. SYSCTRL has privileges allowing operations affecting system resources, but not allowing direct access to data.

Attention: This parameter must be NULL for Windows 95 clients when system security is used (that is, authorization is CLIENT, SERVER, or DCS). This is because Windows 95 does not store group information, thereby providing no way of determining if a user is a member of a designated SYSCTRL group. When a group name is specified, no user is considered to be a member of it and no user is considered to have system control authority. This is not true when DCE authentication is used. In this situation, group names can be specified.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSCTRL_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

System Maintenance Authority Group Name (sysmaint_group)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default Null

Related Parameters

- "System Administration Authority Group Name (sysadm_group)" on page 570
- "System Control Authority Group Name (sysctrl_group)" on page 571

This parameter defines the group name with system maintenance (SYSMAINT) authority. SYSMAINT has privileges to perform maintenance operations on all databases associated with an instance without having direct access to data.

Attention: This parameter must be NULL for Windows 95 clients when system security is used (that is, authorization is CLIENT, SERVER, or DCS). This is because Windows 95 does not store group information, thereby providing no way of determining if a user is a member of a designated SYSMAINT group. When a group name is specified, no user is considered to be a member of it and no user is considered to have system control authority. This is not true when DCE authentication is used. In this situation, group names can be specified.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMAINT_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Authentication Type (authentication)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Client
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] SERVER [CLIENT; SERVER; DCS; DCE]

This parameter determines how and where authentication of a user takes place. If authentication is SERVER, then the user ID and password are sent from the client to the server so authentication can take place on the server. A value of CLIENT indicates that all authentication takes place at the client, so no authentication needs to be performed at the server. For a client-only node, CLIENT, SERVER, and DCS are effectively the same. A value of DCE means that authentication is performed using DCE Security Services. If you are using APPC and a communications product that does not expose the client's password to the DB2 server, you can specify DCS to obtain:

- SERVER-type authentication for non-DRDA clients
- CLIENT-type authentication for DRDA clients

For more information on when and why to use DCE or DCS, see “Authentication” on page 111.

The following show the API constants to use for possible values for this parameter:

- SQL_AUTHENTICATION_SERVER
- SQL_AUTHENTICATION_CLIENT
- SQL_AUTHENTICATION_DCS
- SQL_AUTHENTICATION_DCE

Recommendation: Typically, the default (SERVER) is adequate. If you have incoming requests that are handled by either DB2 Connect or DCE, refer to “Authentication” on page 111.

Default Database Path (dftdbpath)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX	Home directory of instance owner [any existing path]
OS/2 and NT	Drive on which DB2 is installed [any existing path]

This parameter contains the default file path used to create databases under the database manager. If no path is specified when a database is created, the database is created under the path specified by the *dftdbpath* parameter. In a partitioned database environment, you should ensure that the path on which the database is being created is not an NFS-mounted path. The specified path must physically exist on each database partition server. For example, if your `$HOME` directory is locally mounted on one database partition server, while on other database partition servers the directory with the same name is NFS-mounted, then you should not specify `$HOME` as the path for creating the database. To avoid confusion, it is best to specify a path that is locally mounted on each database partition server. The maximum length of the path is 205 characters. The system appends the node name to the end of the path.

Given that databases can grow to a large size and that many users could be creating databases (depending on your environment and intentions), it is often convenient to be able to have all databases created and stored in a specified location. It is also good to be able to isolate databases from other applications and data both for integrity reasons and for ease of backup and recovery.

For UNIX-based environments, the length of the *dftdbpath* name cannot exceed 215 characters and must be a valid, absolute, path name. For OS/2, the *dftdbpath* can be a drive letter, optionally followed by a colon.

Recommendation: If possible, put high volume databases on a different disk than other frequently accessed data, such as the operating system files and the database logs.

LOGON Required for DB2START/DB2STOP (ss_logon)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Database Server with local clients

Parameter Type Configurable

Default [Range] YES [NO (0), YES (1)]

This parameter is applicable to the OS/2 environment only. By accepting the default for this parameter, a LOGON user ID and password is required before issuing a DB2START or DB2STOP.

Trust All Clients (trust_allclnts)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] YES [NO, YES]

Related Parameters

- "Authentication Type (authentication)" on page 572
- "Trusted Clients Authentication (trust_clntauth)"

This parameter is only active when the *authentication* parameter is set to CLIENT.

This parameter and *trust_clntauth* are used to determine where users are validated to the database environment.

By accepting the default of "YES" for this parameter, all clients are treated as trusted clients. This means that the server assumes that a level of security is available at the client and the possibility that users can be validated at the client.

This parameter can only be changed to "NO" if the *authentication* parameter is set to CLIENT. If this parameter is set to "NO," the untrusted clients must provide a userid and password combination when they connect to the server. Untrusted clients are operating system platforms that do not have a security subsystem for authenticating users.

For more information on trusted clients, see "Selecting an Authentication Method for Your Server" on page 113.

Trusted Clients Authentication (trust_clntauth)

Configuration Type Database manager

Applies to

- Database Server with local and remote clients
- Partitioned Database Server with local and remote clients

Parameter Type Configurable

Default [Range] CLIENT [CLIENT, SERVER]

Related Parameters

- "Authentication Type (authentication)" on page 572
- "Trust All Clients (trust_allclnts)" on page 574

This parameter specifies whether a trusted client is authenticated at the server or the client when the client provides a userid and password combination for a connection. This parameter (and *trust_allclnts*) is only active if the *authentication* parameter is set to CLIENT. If a user ID and password are not provided, the client is assumed to have validated the user, and no further validation is performed at the server.

If this parameter is set to "CLIENT" (the default), the trusted client can connect without providing a userid and password combination, and the assumption is that the operating system has already authenticated the user. If it is set to "SERVER," the user ID and password will be validated at the server.

For more information on trusted clients, see “Selecting an Authentication Method for Your Server” on page 113.

Part 5. Appendixes

Appendix A. Planning Database Migration

When you migrate your database, the following events occur:

- The following database entities are migrated:
  - Database configuration file
  - Database system catalog tables
  - Database directories
  - Database log file header
  - Database index files
  - Database data files
- The database is relocated to a new database path.
- System catalog tables are changed as follows:
  - New columns are added.
  - New tables are created.
  - A set of catalog views is migrated, and a set of new catalog views is created, in the SYSCAT schema.
  - A set of updatable catalog views is created in the SYSSTAT schema.
  - A set of general purpose scalar functions is kept, and a set of new general purpose scalar functions is created, in the SYSFUN schema. Only SYSFUN.DIFFERENCE scalar function is dropped and re-created during database migration.
- A new directory called `db2event` is created in the database directory.
- A buffer pool file is created in the database directory.
- A database history file and its shadow are created in the database directory. This file contains a summary of backup information that can be used if a database must be restored, and it is updated whenever a backup, restore, or table load operation is performed on the database. A summary of backup information is also kept for backup and restore operations on a table space.

To plan your database migration to V5, read the following sections:

- Migration Considerations
- Migrating a Database

The details for migrating your database are found in the *Quick Beginnings* manuals for your platform. This appendix will only provide you with an overview of the migration process for planning purposes.

Migration Considerations

To successfully migrate a database created with a previous version of the database manager, you must consider the following:

- Migration Restrictions
- Security and Authorization

- Storage Requirements
- Release-to-Release Incompatibilities

Migration Restrictions

There are certain pre-conditions or restrictions that you should be aware of before attempting to migrate your database to V5:

- Migration is only supported from V1.x or V2.x. Earlier versions of DB2 (Database Manager) must be migrated to V1.x or V2.x before attempting to migrate to V5.
- Issuing the migration command from a V5 client to migrate a database on a V5 Server is supported. However, issuing a migration command from earlier versions of DB2 clients to a V5 Server is not supported.
- Migration between platforms is not supported.
- User objects within your database cannot have V5 reserved schema names as object qualifiers. These reserved schema names include: SYSCAT, SYSSTAT, and SYSFUN.
- Database objects with a dependency on the SYSFUN.DIFFERENCE function must be dropped before migrating the database. Objects that might have a dependency on this function include: views, constraints, functions, and triggers.
- Your database cannot be in one of the following states:
  - Backup pending
  - Roll-forward pending
  - One or more table spaces not in a normal state
  - Transaction inconsistent
- Restore of down-level (V1.x or V2.x) database backups is supported but rolling-forward of down-level logs is not supported.

Security and Authorization

You need SYSADM authority to migrate your database.

If migrating from DB2 Version 1, you should know that a database cannot be cataloged with a mix of authentication types. The authentication type of the instance is, in Version 5, defined in the database manager configuration file. If mixed types are detected during migration from Version 1, you can either stop the migration and change the directories or continue with the migration. If migration continues all the authentication types are changed to blank, and the database uses the authentication type specified in the instance.

To use two databases with different authentication types, a new instance must be created for one of the databases. The database should be backed up and restored to a new database under the new instance. It can then be dropped under the old instance and migration can then be run.

Beginning with Version 2, users and groups are differentiated in SQL statements and the system catalog. As a result, if a user and a group have the same name in the previous version, the authority and privileges granted to the group must be explicitly re-granted after migration.

During migration, the authorization catalog tables, SYSCAT.DBAUTH, SYSCAT.INDEXAUTH, SYSCAT.PLANAUTH, and SYSCAT.TABAUTH, are checked to determine if existing privileges are for users or groups, and the GRANTEETYPE is defined as follows:

- If the name in the GRANTEE column is a user or is not defined; the GRANTEETYPE is defined as U.
- If the name in the GRANTEE column is a group; the GRANTEETYPE is defined as G.
- If the name in the GRANTEE column is both a user and a group; the GRANTEETYPE is defined as U. Privileges must then be explicitly granted to the group.

Storage Requirements

Space is required for both the old and new catalogs during the migration, and the amount of disk space required will vary depending on the complexity of the database as well as the number and size of the database objects. These objects include all tables and views. You should make available at least two times the amount of disk space as the database catalog currently occupies. If there is not enough disk space, migration fails.

You should also consider increasing the database configuration parameters associated with the log files. You should increase `logfilsiz`, `logprimary`, and `logsecond` to prevent the space for these files from running out. If log space is completely used, you will receive a SQLCODE of SQL1704N with a reason code of 3. If this happens, increase the log space parameters and re-issue the database migration command.

Release-to-Release Incompatibilities

To successfully migrate a database, you should consider the impact of the incompatibilities between the two versions of the product. The following incompatibilities deserve special attention before you begin your migration:

- **View Definitions**

  If a view from before Version 2 involves “SELECT *,” the view may be unusable after migration. If the view is unusable, attempts to use it, directly or indirectly, will result in SQLCODE -158. The view must be dropped and recreated in order to avoid this error. If fewer than the current number of columns in the SELECT * table is desired, the recreated view must specify the needed columns.

- **Configuration Parameters**

  Configuration parameter values are preserved during the migration of the database, with the exception of the following parameters:

  – Application Heap Size (applheapsz)

– Package Cache Size (pckcachesz) (applies to all platforms except DB2 for OS/2 V1.x.)
– Maximum Storage for Lock List (locklist)
– Recovery Range and Soft Checkpoint Interval (softmax).

For these parameters, the use of the associated heap has changed significantly in Version 5.

– *applheapsz* is reset to the Version 5 default value if the current value is less than the Version 5 default value.
– *pckcachesz* is always reset to the Version 5 default value.

For *locklist*, the DB2 Version 1 or DB2 Parallel Edition Version 1.2 value is multiplied by a factor of 32/25. This computed value will be used as the Version 5 parameter value, if this value is greater than the Version 5 default. Otherwise, the Version 5 default will be used.

OS/2 Users: If you are migrating from Version 1, parameters previously allocated in units of 64 KB *segments* are multiplied by 16 to allow for allocation in units of 4 KB pages. In addition, the *softmax* configuration parameter will be set to the default value, since this parameter is now measured as a percentage of the log records written rather than the number of log records written.

In order to take advantage of Version 5 enhancements, you should re-tune your database manager and database configuration after migrating your databases. To assist in this tuning, you may wish to record and compare configuration parameter values from before and after your migration. (See the GET DATABASE CONFIGURATION and GET DATABASE MANAGER CONFIGURATION commands in the *Command Reference* manual.)

Migrating a Database

The following are the steps you must take to migrate your database. The database manager must be started before migration can begin.

PRE-MIGRATION:

Note: The pre-migration steps must be done on a previous release (that is, on your current release before migrating to, or installing, the new release).

1. You cannot migrate a database that is in one of the following states:

   - Backup pending
   - Roll-forward pending
   - One or more table spaces not in a normal state
   - Database inconsistent

   You cannot migrate a database that contains any database objects with a dependency on scalar function SYSFUN.DIFFERENCE.

   In addition, you cannot migrate a database that contains any database objects which have a qualifier (schema name) of SYSCAT, SYSSTAT, and SYSFUN. These schema names are reserved for use by the database manager.

See the *Quick Beginnings* for information about migrating from previous releases, and for information about functions to help with the above step of the migration process. This book also introduces when and how to use the DB2CKMIG pre-migration utility.

2. All applications and end users must be disconnected from each database being migrated. Use the LIST APPLICATIONS and the FORCE APPLICATIONS commands as necessary.
3. Use the DB2CKMIG pre-migration utility presented in the *Quick Beginnings* for your platform to check to see if the database can be migrated. Re-use the utility until there are no more errors. Typical corrections include:
   - Drop and re-create objects using valid schema names.
   - Correct database connection states.
   - Remove all dependencies from objects on scalar function SYSFUN.DIFFERENCE.
4. Backup your database.

   Migration is not a recoverable process. If you backup your database before the Version 5 restricted schema names are changed, you will not be able to restore the database from backup using DB2 Version 5. To restore the database, you will have to use the version of the database manager from which you are migrating your databases.

   Attention! If you do not have a backup of your database from before you attempted migration, and the migration failed, you will have no way of restoring your database using DB2 V5 or your previous version of the database manager.

   You should also be aware that any database transactions done during the period between the time the backup was completed and the time the upgrade to V5 is complete are not recoverable. That is, if sometime following the completion of the installation and migration to V5, the database needs to be restored (to a V5 level), the logs from before V5 installation cannot be used in roll-forward recovery.

MIGRATION:

5. Migrate the database using one of the following:
   - The SQLEMGDB migrate database API
   - The MIGRATE DATABASE command-line processor command
   - The RESTORE DATABASE command, when restoring a full backup of the database.

     Note: To restore a Version 1 OS/2 database backup, you must use DB2RESDB.

   OS/2 Users: The DB2CIDMG migration program, which works in a Configuration/Installation/Distribution (CID) architecture environment, is only available on DB2 for OS/2. It allows for remote unattended installation and configuration on LAN-based workstations. You must have NetView DM/2 on your LAN to use CID migration.

UNIX Users: The *Quick Beginnings* describes what to do if you do not want to migrate all databases in a given instance.

Note: During installation of V5, all of the found local database directories are migrated. It may be that you require keeping one of your current local database directories past the time of the installation of Version 5. (For example, your operating system may allow a dual boot feature: where you can have the original version of DB2 when “booting”your system one way, and the new version when “booting” the other way.) If you keep your current directories, then you may need a way to migrate that database directory to the Version 5 format at some later time. The DB2MIGDR utility allows you to complete this migration.

POST MIGRATION:

6. Optionally, use the DB2UIDDL utility to assist in searching all unique indexes from the migrated database. This utility creates a file containing a list of CREATE UNIQUE INDEX statements. Executing this file as a DB2 CLP command file results in the unique index being converted to Version 5 semantics. Refer to the *Quick Beginnings* manuals for more details.
7. Optionally, issue RUNSTATS on tables that are particularly critical to performance of SQL queries. Old statistics form the previous level database are retained in the migrated database. Therefore, any new statistics that are modified for, or are new to, Version 5 will not be added to the migrated database unless you issue RUNSTATS.
8. Optionally, rebind all packages. If migrating from a Version 2 database, there may be inoperative packages. Inoperative packages remain identified as inoperative following migration. All existing valid packages are marked as invalid during catalog migration. You can use the DB2RBIND utility to revalidate all packages, or allow package revalidation to occur implicitly when a package is first used. The REBIND PACKAGE or BIND commands will selectively bind a particular package
9. Tune your database and database manager configuration parameters to take advantage of Version 5 enhancements.
10. Optionally, migrate Explain tables if you have been using the Explain tables and are planning to use them in Version 5. There are several new columns in the tables. See Chapter 13, “SQL Explain Facility” on page 377 and Appendix K, “ Explain Tables and Definitions” on page 763 for more information.

Complete details on the migration steps are found in the *Quick Beginnings* manuals for your platform.

Appendix B. Incompatibilities Between Releases

This appendix identifies the incompatibilities that exist between DB2 Universal Database and previous releases of DB2.

An “incompatibility” is defined to be a part of DB2 Universal Database that works differently than it did in a previous release of DB2 in such a way that if it used in an existing application it will produce a different result, necessitate a change to the application, or reduce performance. In this definition, “application” can apply to a broad range of things, such as:

- Application program code
- Third-party utilities
- Interactive SQL queries
- Command and/or API invocation.

This appendix does not describe incompatibilities where certain operations in the current release are less likely to generate an error condition than they did in the previous release, as those changes will only have a positive impact on existing applications.

This appendix lists incompatibilities in the following categories:

Each incompatibility includes a description of the change in DB2 Version 5 that causes an incompatibility with previous releases, the symptom or effect this will have on your environment if no changes are made to it, and the possible resolutions that are available. There is also an indicator at the beginning of each incompatibility telling you what platforms are applicable as follows:

DB2 PE DB2 Parallel Edition, Version 1.2

OS/2 OS/2

UNIX Unix-based operating systems supported by DB2

WIN Microsoft Windows platforms supported by DB2

System Catalog Tables/Views

System Catalog Views

UNIX	OS/2	WIN	DB2 PE

Change

A set of views have been created in DB2 Version 2 with the qualifiers (also known as *schema names*) of SYSCAT and SYSSTAT. For this reason, the SYSCAT and SYSSTAT schemas are now reserved.

Symptom

If there are any objects belonging to these schemas in a Version 1 database, migration will fail with SQLCODE SQL1704N (reason code 1).

Resolution

The only way to get through the migration successfully will be to recreate the objects currently under the SYSCAT and SYSSTAT schemas under new high level qualifiers.

System Catalog Tables

UNIX	OS/2	WIN

Change

A variety of changes have been made to the SYSIBM tables. This section will discuss the subset which could cause incompatibilities. To see a description of all changes (for example, new columns, new values in a column, and so on) refer to the *SQL Reference*.

SYSCOLUMNS

COLTYPE: Changed values: “FLOAT” to “DOUBLE”

NULLS: Changed values: “D” to “N”. (Default flag now found in SYSCAT.COLUMNS.DEFAULT)

HIGH2KEY: Changed type: VARCHAR(16) to VARCHAR(33).
Changed values: Values now stored in printable format rather than binary format

LOW2KEY: Changed type: VARCHAR(16) to VARCHAR(33).
Changed values: Values are now stored in printable format rather than binary format for all datatypes.

SYSINDEXES

CLUSTERRATIO: Changed value: Value will always be -1 if the columns CLUSTERFACTOR and PAGE_FETCH_PAIRS are populated.

SECT_INFO: Changed type: LONG VARCHAR to BLOB(1M).

HOST_VARS: Changed type: LONG VARCHAR to BLOB(1M).

ISOLATION: Changed type: CHAR(1) to CHAR(2). Changed values: “R” to “RR”, “S” to “RS”, “C” to “CS”, “U” to “UR”.

SYSRELS

RELNAME: Changed type: CHAR(8) to VARCHAR(18).

SYSSECTION

SECTION: Changed type: VARCHAR(3900) to VARCHAR(3600)

SYSSTMT

TEXT: Changed type: VARCHAR(3900) to VARCHAR(3600)

SYSTABLES

PACKED_DESC: Changed type: LONG VARCHAR to BLOB(10M)
VIEW_DESC: Changed type: LONG VARCHAR to BLOB(32K)
REL_DESC Changed type: LONG VARCHAR to BLOB(32K)
FID Will no longer uniquely identify a table on its own. Must be used with TID to uniquely identify a table.

SYSVIEWS

CHECK: Changed values: "Y" to "L".

TEXT: Changed type: VARCHAR(3900) to VARCHAR(3600). Contains the full text of the create view statement (including the CREATE VIEW). In Version 1, only the select portion was shown.

Symptom

A variety of symptoms could occur.

If you have an application which has a qualified search on a field that has had a value change (for example, ISOLATION in SYSIBM.SYSPLAN) this will cause your application to react differently than you would want.

If you have an application which accesses some field where the field type or size has changed (such as SECTION in SYSIBM.SYSSECTION), you could retrieve an incomplete set of data, too much data, or have the wrong type defined in your application to represent the data type of the table column.

Resolution

If you use the SYSIBM tables for application processing or anything else, you must review the changes listed above to decide whether or not you are affected and what the appropriate action to correct the situation is. You may need to refer to the *SQL Reference* to understand what the new columns, new values for columns and other changes that were made to these tables.

If you need a rough approximation of the degree of clustering, select both CLUSTERRATIO and CLUSTERFACTOR and choose the “greater” one.

Unique Table Identification

UNIX	OS/2	WIN

Change

With the introduction of table spaces, the TID column of SYSIBM.SYSTABLES is now used to identify a table space. The FID column of SYSIBM.SYSTABLES will no longer uniquely identify a table is a database. FID now uniquely identifies a table in a table space. This means that to uniquely identify a table in a database you need the combination of TID and FID.

Symptom

Any application which assumes FID will uniquely identify a table in a database may process incorrectly should the FID be duplicated in multiple table spaces.

Resolution

Change the application to use TBSPACEID and TABLEID from the SYSCAT.TABLES view as the unique identifier. You can also use the columns TID and FID from SYSIBM.SYSTABLES.

Application Programming

NS and NX Lock Modes

UNIX	OS/2	WIN	DB2 PE

Change

Due to the addition of NS and NX lock modes in DB2 Version 5, there is a difference in the behaviour of index scans with isolation level Cursor Stability (CS) or Read Stability (RS).

Symptom

In DB2 Version 5, an index scan with isolation level, CS or RS, will not see an uncommited delete of a row that is within the scanned range. In DB2 Version 2, the scanner would not see an uncommitted delete of a row that was at the end of the scanned range. However, if the deleted row was within the range, the scanner would remain in a lock wait until the delete was committed or rolled back.

For example in DB2 Version 5, the following can occur with an index on Column A:

Sequence	Application 1	Application 2
1	delete from t1 where a=3	
2		select a from t1 where a>1 and a<5 A ---------- 2 4 5
3	rollback	
4		select a from t1 where a>1 and a<5 A ---------- 2 3 4 5

The same scenario in previous versions of DB2 would result in application 2 being in lock wait until Application 1 committed or rolled back.

Resolution
There is no resolution as this is an enhancement to isolation level Cursor Stability or Read Stability.

CREATE TABLE NOT LOGGED INITIALLY

UNIX	OS/2	WIN	DB2 PE

Change
In DB2 PE V1.2, in the unit of work in which a table is created with the NOT LOGGED INITIALLY option, an error on this table will cause the unit of work to be rolled back. In Version 5, the range of errors that will cause a roll back has been increased.

Symptom
In Version 5, in the unit of work in which a table is created with the NOT LOGGED INITIALLY option, an error in any operation involving any table will cause the unit of work to be rolled back.

Resolution
Correct the error and run the transaction again.

DB2 Call Level Interface (DB2 CLI) Defaults

UNIX	OS/2	WIN

Change
The default values for **AUTOCOMMIT** and **CURSORHOLD** have changed. Both AUTOCOMMIT and CURSORHOLD will now default to ON.

Symptom
If an application was written assuming that AUTOCOMMIT was OFF or that WITH HOLD semantics was NOT used for cursors, then these default changes could cause the application to fail.

Resolution
Add one or both of the following two lines to your DB2CLI.INI file.

- AUTOCOMMIT = 0
- CURSORHOLD = 0

Obsolete DB2 CLI Keywords

UNIX	OS/2	WIN

Change
You can control DB2 configurable features by specifying a set of optional keywords in an DB2 CLI initialization file. In DB2 Version 2, some of these keywords become obsolete, as follows:

1. UNDERSCORE
2. TRANSLATEDLL
3. TRANSLATIONOPTION

Symptom
These keywords will be ignored if they still exist. You may notice behavioral changes based on the removal of these settings.

Resolution
You will need to review the new list of valid parameters to decide what the appropriate keywords and settings are for your environment. See the *CLI Guide and Reference* for information on these keywords.

DB2 CLI SQLSTATEs

UNIX	OS/2	WIN

Change
A more explicit set of SQLSTATEs (in the S1090 to S1110 range) has replaced the generic SQLSTATE S1009.

Symptom
SQLSTATE values returned to the application calling DB2 CLI APIs have changed.

Resolution
Update your application to check for the new SQLSTATEs. Refer to the *Message Reference* for a complete list of these SQLSTATEs.

DB2 CLI Mixing Embedded SQL, Without CONNECT RESET

UNIX	OS/2	WIN

Change

DB2 CLI's Version 2 support of multiple connections may affect your existing applications. If your application connects to a database using any non-CLI interface (including embedded SQL using the command line processor or administrative APIs) and does NOT issue a reset before connecting to a database using DB2 CLI, your applications will be affected by this change.

Symptom

The second connect will fail with an SQLSTATE of 08001 since it is not same type of connection as the first connect.

Resolution

The application must issue a CONNECT RESET before calling a DB2 CLI connect function.

DB2 CLI Use of VARCHAR FOR BIT DATA

UNIX	OS/2	WIN

Change

Character data defined with the FOR BIT DATA clause is now by default mapped to the new C buffer type, SQL_C_BINARY. If data is defined as FOR BIT DATA, it is transferred to:

- SQL_C_BINARY buffers unchanged
- SQL_C_CHAR buffers as a character representation of the hexadecimal value of the data. Each byte is represented by two ASCII characters, (meaning the SQL_C_CHAR buffer must be double the size of the FOR BIT DATA string.)

Symptom

Existing applications that explicitly use SQL_C_CHAR with data defined as FOR BIT DATA, will get a different result and may receive only half of the original data.

Resolution

In order to have DB2 CLI treat FOR BIT DATA the same as it did in Version 1, add the following line to DB2CLI.INI:

```
BITDATA = 0
```

DB2 CLI Data Conversion Values for SQLGetInfo

UNIX	OS/2	WIN

Change
The SQL_CONVERT_xxxx *fInfoType* is defined by ODBC to indicate supported conversion functions. A change has been made in how we handle SQL_CONVERT_xxxx *fInfoType*s which were used with the corresponding SQL_CVT_xxx comparison masks to correctly follow ODBC standards.

Symptom
DB2 CLI will no longer return bit masks for the SLQ_CONVERT_xxx *fInfoType*s and corresponding SQL_CVT_xxx comparison masks. DB2 CLI Version 2 now returns zero for all SQL_CONVERT_xxx *fInfoType*s.

Resolution
This is to correct Version 1 processing which was not ODBC compliant. There is no resolution.

DB2 CLI/ODBC Configuration Keyword Defaults

UNIX	OS/2	WIN

Change
The default value for the CLI/ODBC configuration keyword DEFERREDPREPARE has changed. In DB2 CLI Version 5 deferred prepare is now on by default.

Symptom
Applications that rely on the prepare to be executed as soon as it is issued will not function as expected. In particular, the row and cost estimates normally returned in the SQLERRD(3) and SQLERRD(4) of the SQLCA of a prepare statement may become zeros. The application will not be able to use this information to decide whether or not to continue the execution of the SQL statement.

Resolution
Add the following line to your *db2cli.ini* file:

```
DEFERREDPREPARE = 0
```

Obsolete DB2 CLI/ODBC Configuration Keywords

UNIX	OS/2	WIN

Change
You can change the behavior of the DB2 CLI/ODBC driver by specifying a set of optional keywords in the *db2cli.ini* file. In Version 5, the AUTOCOMMIT keyword has become obsolete.

Symptom
These keywords will be ignored if they still exist. You may notice behavioral changes based on the removal of these settings.

Resolution
You will need to review the new list of valid parameters to decide what the appropriate keywords and settings are for your environment. See the *CLI Guide and Reference* for information on these keywords.

DB2 CLI SQLSTATEs

UNIX	OS/2	WIN

Change
The category of SQLSTATEs that started with S1 in DB2 CLI Version 2 have been renamed to begin with HY in Version 5. For example, the SQLSTATE S1010 is now HY010.

Symptom
SQLSTATE values returned to the application calling DB2 CLI APIs have changed.

Resolution
Applications should be updated to expect the new HY class of SQLSTATEs. Alternatively, the environment attribute SQL_ATTR_ODBC_VERSION can be set to SQL_OV_ODBC2 using the DB2 CLI function `SQLSetEnvAttr()`. The DB2 CLI/ODBC driver will then return the S1 class of SQLSTATEs.

Stored Procedure Catalog Table

UNIX	OS/2	WIN

Change
Version 5 now has 2 system catalog views used to store information about all the stored procedures on the server (SYSCAT.PROCEDURES and SYSCAT.PROCPARMS). These replace the Version 2 pseudo catalog table for stored procedures

Symptom
By default the server will look in the new system catalog views for information about stored procedures, not the older pseudo catalog table. DB2 CLI functions such as `SQLProcedureColumns()` and `SQLProcedures()` will therefore not return the appropriate information.

Resolution
Register the stored procedures using the CREATE PROCEDURE SQL command. See the *SQL Reference* for more information. Alternatively, the DB2 CLI/ODBC configuration keyword PATCH1 can be set to 262144 to force the DB2 CLI/ODBC driver to use the pseudo catalog table as it did in Version 2.

PREP Command - LANGLEVEL

UNIX	OS/2	WIN

Change
When the LANGLEVEL MIA option of the PREP command is used, all C null-terminated strings are padded with blanks and the null-terminating character is placed in the last byte of the string.

Symptom
Although this change was made for MIA compliance, it has caused a change to the way C null-terminated strings are handled.

Resolution
There is another LANGLEVEL setting (SAA1) which may cause the behavior to better match your needs. You should review the options and decide what is best for your environment.

Change to SMALLINT Constants

UNIX	OS/2	WIN

Change
Integer constants in the range -32,768 to 32,767 are now treated as INTEGER types, rather than SMALLINT. This resolves an incompatibility with IBM SQL, as well as simplifying the rules for determining literal types.

It is also worth mentioning that the smallest INTEGER constant in Version 1 (-2147483638) is a DECIMAL constant with precision 10 and scale 0 in Version 5.

Symptom
This affects the result precision and scale of decimal operations. (Which impacts, for example, the print width of decimal fields.)

Resolution
There is no resolution. This change in handling integers results in an increase in precision.

Error Handling

UNIX	OS/2	WIN

Change
Errors which were previously reported at bind time may now not occur until statement execution. For instance, if you create a table using incorrect SQL syntax such as:

```
CREATE TABLE T1 (C1 CHAR(5), C1 CHAR(10))
```

The error that a duplicate column name was used will be flagged at run time instead of bind time. For all DDL statements, syntax errors are reported at bind time and semantic errors are reported at run time.

Symptom

Some errors which were reported at bind time in Version 1 will now be reported at execution time.

Resolution

As long as the application has proper error handling routines, this should not cause a problem. There will be some additional errors which can now occur during execution.

Maximum Number of Sections in a Package

UNIX	OS/2	WIN

Change

The limit for the maximum number of sections in a package has changed from 400 to whatever the storage allows. This limit used to be hard-coded at 400, but now depends on the type of SQL statements in the program. As a result of this change, the constant for the maximum number of SQL statements has been removed from the common include files *sql.h*, *sql.cbl*, and *sql.f*.

Symptom

If an application program references the following constants, it will not compile successfully in Version 5:

- SQL_MAXSTMTS (in *sql.h*)
- SQL-MAXSTMTS (in *sql.cbl*)
- SQL_MAXSTMTS (in *sql.f*)

Resolution

Remove references to these constants or define them directly within your application.

Bind Warnings

UNIX	OS/2	WIN

Change

Version 1 reports a warning at bind time if the number of host variables in an INTO clause is less than the number of select list items. Version 2 reports the same bind time warning if there are more or less host variables than select list items.

Symptom

You will receive bind time warning messages where one was not received in Version 1.

Resolution
Rebind the application with the new bind option SQLWARN NO and warnings will not be reported.

Bind Options

UNIX	OS/2	WIN

Change
The new SQLWARN bind option has a default value of 'YES'.

Symptom
By default, positive SQLCODEs may now be returned on DESCRIBE, PREPARE, and EXECUTE IMMEDIATE requests which were previously not returned. (For instance, a SQLCODE of +236 may be returned).

Resolution
Rebind with SQLWARN NO if your application cannot tolerate positive SQLCODEs or treats them as errors.

PREP with BINDFILE

UNIX	OS/2	WIN

Change
In Version 2, under certain circumstances, the DB2 PREP (precompile) command allows a bind file to be created even if certain errors occur. If the BINDFILE option, but not the PACKAGE option, is specified on the prep command, the following object existence and authority errors will be tolerated:

SQL0117N The number of values assigned is not the same as the number of specified or implied columns.

SQL0204N "<name>" is an undefined name.

SQL0205N "<name>" is not a column of table "<table-name>".

SQL0206N "<name>" is not a column in an inserted table, updated table, or any table identified in a FROM clause or is not a valid transition variable for the subject table of a trigger.

SQL0440N No function by the name "<function-name>" having compatible arguments was found in the function path.

SQL0551N "<authorization-ID>" does not have the privilege to perform operation "<operation>" on object "<name>".

SQL0552N "<authorization-ID>" does not have the privilege to perform operation "<operation>".

Symptom

This may cause precompilation of some applications to succeed with errors where they failed in previous versions. The resultant bind file will fail if it is bound to a database with similar omissions of objects or authorities.

Resolution

Check bind errors instead of precompiler errors for this condition.

Varchar Structures in COBOL

UNIX	OS/2	WIN

Change

The COBOL precompiler in Version 2 and Version 5 supports declaration of group data items as host variables. (Refer to the *Embedded SQL Programming Guide* for more information.) This may cause some incompatibility with existing applications which did not adhere to the precise declaration format for VARCHAR host variables in COBOL.

The level numbers for subordinate items, as documented in DB2 manuals, must be 49. The following declaration would be accepted by the COBOL precompiler in Version 1, but not in Version 2 or Version 5:

```
01 MY-VAR.
     10 MY-LENGTH PIC S9(4) COMP-5.
     10 MY-DATA   PIC X(100).
```

If not coded correctly, the Version 2 and Version 5 precompiler will treat declarations like the above as structures with two members, a short integer and a fixed-length character string. When such a variable is used in an SQL statement, the reference to the would-be VARCHAR would be replaced with references to the two subordinates.

Symptom

Depending on the situation, this may result in the following message:

```
SQL0087N Host variable "<name>" is a structure used where
         structure references are not permitted.
```

Resolution

Applications being migrated to Version 5 that contain host variables which are intended to be VARCHARs should be declared with the subordinates at level 49.

Incompatible APIs

UNIX	OS/2	

Change

Several APIs have been changed or removed since Version 1. See the charts in the *API Reference* showing descriptions of the changes.

Symptom
In most cases, the original API call will still work, however, you cannot take advantage of any of the new Version 5 functionality while using the old API calls or parameters.

Resolution
Applications should be upgraded to use the new Version 5 API calls as described in the *API Reference*.

Supported Level of JDBC

UNIX	OS/2	WIN

Change
The supported level of the JDBC (Java Database Connectivity) API has changed. DB2 Version 5 provides a driver for JDBC 1.1 instead of JDBC 1.0, which came with DB2 Version 2.1.2.

Symptom
Compiled JDBC 1.0 clients fail when executed directly as a DB2 Version 5 client. Old Java classes are not found.

Resolution
To continue using JDBC 1.0 clients, run them on a DB2 Version 2.1.2 client, with a remote DB2 Version 5 server. Modify the client source code to upgrade to the JDBC 1.1 API. Run the JDBC 1.1 clients on a Java Development Kit Version 1.1-compatible virtual machine.

Calling Convention for Java Stored Procedures and UDFs

UNIX	OS/2	WIN

Change
The calling convention for Java stored procedures and user-defined functions (UDFs) has changed in DB2 Version 5.

Symptom
Java stored procedures and UDFs written for DB2 Version 2.1.2 will not be found when run on DB2 Version 5.

Resolution
Change the Java stored procedure and UDF source code to use the new calling convention. Refer to the *Embedded SQL Programming Guide* for details.

Java Runtime Environment

UNIX	OS/2	WIN

Change
The level of the Java runtime environment required for Java stored procedures, user-defined functions, and JDBC clients has changed in DB2 Version 5.

Symptom
The JDBC DLL will not load when JDBC 1.1 clients are run. Java stored procedures and UDFs will fail.

Resolution
Install a compatible Java 1.1 runtime environment at the client and server. At the server, set the `jdk11_path` configuration parameter.

Obsolete System Monitor Requests for DB2 PE Version 1.2

			DB2 PE

Change
Some request types that were available with the DB2 PE Version 1.2 system monitor are no longer supported. See the tables in the *System Monitor Guide and Reference* showing descriptions of the changes.

Symptom
The old request types will not work.

Resolution
Applications should be upgraded to use the new DB2 Version 5 requests types as described in the *System Monitor Guide and Reference*.

SQL

Updating Partitioning Key Columns

UNIX	OS/2	WIN	DB2 PE

Change
In DB2 PE Version 1.2, partitioning key columns could be updated if the table was in a single-node nodegroup. In DB2 Version 5, partitioning key columns can be updated if the table is in a table space in a single-node nodegroup, and there is no partitioning key defined.

Symptom
An update statement fails with SQL270N (SQLCODE -270, SQLSTATE 42997) with reason code 2. The same error is returned if the table is in a table space in a single or multiple node nodegroup.

Resolution
If the table is in a table space in a single node nodegroup, then use the ALTER TABLE statement to DROP the partitioning key. As with DB2 PE Version 1.2, if the table is in a table space in a multiple node nodegroup, the nodegroup must be changed to a single-node nodegroup and REDISTRIBUTE NODEGROUP must be issued before attempting to update partitioning key columns.

Column NGNAME

UNIX	OS/2	WIN	DB2 PE

Change
In DB2 PE Version 1.2, a table was directly associated with a nodegroup. In DB2 Version 5, a table is in a table space, which is within a nodegroup. Since there is no longer a direct relationship with a nodegroup, there is no need for a column, named NGNAME in the SYSIBM.SYSTABLES catalog table.

Symptom
An SQL statement that refers to the NGNAME column from SYSIBM.SYSTABLES catalog table will return an SQLCODE of –206 (SQLSTATE 42703).

Resolution
Remove the column NGNAME from the SQL statement. To determine the nodegroup name for the table, refer to NGNAME in the row of SYSCAT.TABLESPACES catalog view, that relates to the table space in which the table is stored.

Node Number Temporary Space Usage

UNIX	OS/2	WIN	DB2 PE

Change
When using a temporary table that requires row identifiers, the amount of space needed is increased to include the node number. The space limit for temporary tables is 4005 bytes. If temporary tables are close to the 4005 byte limit, any further increase can exceed this limit.

Symptom
There are two possible symptoms of this change.

- An SQL statement may fail to compile and return an SQLCODE of SQL0670N (SQLSTATE 54010).
- The temporary table is not used which may effect the performance of the query.

Resolution
You should review and use the directions in the Actions section of the message details for SQL0670N to fix the error.

Authorities for Create and Drop Nodegroups

UNIX	OS/2	WIN	DB2 PE

Change
The authorization required for creating or dropping a nodegroup has changed from SYSADM or DBADM to SYSADM or SYSCTRL. This means that a user ID with DBADM authority cannot create, alter, or drop nodegroups.

Symptom
A user ID, with DBADM authority, issuing a CREATE NODEGROUP or DROP NODEGROUP statement will receive an SQL00551N (SQLSTATE 42501).

Resolution
Issue the statement using a user ID that has SYSADM or SYSCTRL authority. For your convenience, you may wish to include the user ID in the SYSCTRL group. Refer to the *Administration Guide* for further information.

Target Map in REDISTRIBUTE NODEGROUP

UNIX	OS/2	WIN	DB2 PE

Change
The specification of a target map in the REDISTRIBUTE NODEGROUP command or API no longer causes database partitions to be implicitly added or dropped from the node group. This means that the target map cannot include nodes that are not defined to the node group. An undefined node that is included in the target map file will cause an error to be returned. A database partition, which has been defined to the node group, can be excluded from the target map file and will not appear in the partition map.

Symptom
If a node is included in the target map file and was not defined to the node group, the REDISTRIBUTE NODEGROUP command will return an SQLCODE–6053 with a reason code 6.

Resolution
Before issuing the REDISTRIBUTE NODEGROUP command, add the database partition to the node group, using the ALTER NODEGROUP statement. You can also drop the node from the node group using the ALTER NODEGROUP statement, either before or after issuing the REDISTRIBUTE NODEGROUP command. Refer to the *SQL Reference* for further information on the ALTER NODEGROUP statement.

Node Group for Create Table

UNIX	OS/2	WIN	DB2 PE

Change

In DB2 PE Version 1, a table was directly associated with a node group. In DB2 Version 5, a table is in a table space within a node group. When a user issues a CREATE TABLE statement, the name following the IN keyword is a table space name, not a node group name. The default table space selected may not be defined in the IBMDEFAULTGROUP node group, which was the default node group in DB2 PE Version 1.

Symptom

If you use existing CREATE TABLE statements from DB2 PE Version 1, they may fail with an SQLCODE of SQL0204N (SQLSTATE 42704), with the name specified following the IN keyword in the message. This will occur if a table space with the same name as the node group has not been automatically created during database migration.

If you are using CREATE TABLE statements that do not specify the IN keyword, the table space selected, by default, may not be using the node group, IBMDEFAULTGROUP, and will not include data on all the database nodes. You can check the partition map for the table to confirm this.

Resolution

Ensure that any name specified following the IN keyword on the CREATE TABLE statement is the name of a defined table space. For existing statements, you could set up a table space for each node group with the same name.

To ensure that tables default to the IBMDEFAULTGROUP for all users, define a table space called IBMDEFAULTGROUP, defined in the node group, IBMDEFAULTGROUP. This ensures that tables created by any users will default to use this table space.

Note: This is done automatically during database migration from DB2 PE Version 1 to DB2 Version 5.

Revoking CONTROL on Tables or Views

UNIX	OS/2	WIN	DB2 PE

Change

A user can grant privileges on a table or view using the CONTROL privilege. In DB2 Version 5, the WITH GRANT OPTION provides a mechanism to determine a user's authorization to grant privileges on tables and views to other users. This mechanism is used in place of CONTROL to determine whether a user may grant privileges to others. When CONTROL is revoked, users will continue to be able to grant privileges to others.

Symptom
A user can still grant privileges on tables or views, following the revocation of CONTROL privilege.

Resolution
If a user should no longer be authorized to grant privileges on tables or views to others, revoke all privileges on the table or view and grant only those required.

High Level Qualifiers for Objects in DB2 Version 5

UNIX	OS/2	WIN	DB2 PE

Change
In DB2 PE Version 1, users would create a table, view, index or package with any schema name or qualifier with the exception of SYSIBM. This differs from other IBM database products and is not compliant with SQL92. In DB2 Version 5, there are limits of the schema names that you can use.

- The schema names for tables, views, indexes, and packages cannot be SYSIBM, SYSCAT, SYSSTAT, OR SYSFUN.

  Note: The schema names for all other objects must not start with SYS.

- Each schema is an object defined in the database catalog.

  Users require IMPLICIT_SCHEMA authority to implicitly create a schema. Once a schema is created, specific privileges allow users to create objects (CREATEIN privilege), drop any object in the schema (DROPIN privilege), or alter (comment on) any object in the schema (ALTERIN privilege). The change to supporting schemas, as objects with privileges, has resulted in changes to privileges associated with various statements.

  – For creating objects in an existing schemas, you must have CREATEIN privilege.
  – For creating objects in a schema that does not exist, you must have IMPLICIT_SCHEMA authority.
  – For dropping objects in a schema, you must be the definer of the object, have CONTROL privilege, or have DROPIN privilege on the schema.
  – For altering, including commenting on, objects in a schema you must be the definer of the object, have CONTROL privilege, or have ALTERIN privilege on the schema.

    Note: For altering or commenting on a table, the ALTER privilege on the table is also valid.

Symptom
If you create an object with an invalid schema name, the CREATE statement returns an SQLCODE of SQL0553N. This message indicates that the object cannot be created with the schema name.

If a CREATE, ALTER, COMMENT ON or DROP statement returns an SQLCODE of SQL0551N, you did not have the necessary privilege. This may be the result of schema-related privileges and could indicate that:

- The object cannot be created because the schema does exist and you do not have the IMPLICIT_SCHEMA authority.
- The object cannot be created because the schema does not exist and you do not have the CREATEIN privilege.
- The object cannot be dropped because another user created the object and you do not have the DROPIN privilege.
- The object cannot be altered (commented on) because another user created the object and you do not have ALTERIN privilege.

Resolution

Depending on the symptom:

- Do not create schema names with SYS.
- If a user can create a table, view, index or package, grant the necessary authority or privilege using the GRANT (Database Authorities) statement for IMPLICIT_SCHEMA authority, or the GRANT (Schema Privileges) statement for CREATEIN, DROPIN or ALTERIN privilege on the schema. A user with DBADM authority must first create the schema.

Inoperative VIEWs

UNIX	OS/2	WIN

Change

In DB2 Version 2, a view is made inoperative if a SELECT privilege upon which the view definition is dependent is revoked or if an object upon which the view definition is dependent was dropped (or possibly made inoperative in the case of another view). This is in contrast to the behavior in DB2 Version 1 where the view would have been dropped under the same circumstances.

Symptom

If the use of an inoperative VIEW is attempted, an SQL0575N will be returned to the application.

Resolution

To resolve this problem, you will need to do two things:

1. Resolve the dependency (such as CREATE the dropped table).
2. Execute a CREATE VIEW.

   Since the view is only inoperative and not dropped, you can query the TEXT column of SYSCAT.VIEWS to retrieve the current definition of the view.

Unusable VIEWs

UNIX	OS/2	WIN

Change
If you currently have a view defined with SELECT * on a table as part of the view definition, the view may be unusable after migration.

Symptom
You will receive an SQL0158N error if you attempt to use a view that is unusable.

Resolution
In order to resolve this problem you will need to:

1. Drop the existing view (DROP VIEW command).
2. Re-create the view (CREATE VIEW command), specifying column names in place of "*".

SQLCODE Changes

UNIX	OS/2	WIN

Change
The SQLCODEs returned for an INSERT or UPDATE statement resulting in data being out of range have changed. These are:

- SQL0406N is now SQL0413N
- SQL0404N is now SQL0433N

The message has changed from "A numeric value/string in the UPDATE or INSERT statement is ..." to "Overflow occurred during numeric data type conversion". Note that there have been no changes to the corresponding SQLSTATEs.

Symptom
These SQLCODEs are caused by trying to place a value in a column that is outside a limit that exists on the data in that column. For applications, different values will now be returned in SQLCA.SQLCODE. In any interactive situation (such as using the command line processor), a different error code will be reported to the user.

Resolution
If your application specifically looks for the old SQLCODEs, you will need to change the comparison to use the new codes.

WITH CHECK OPTION on CREATE VIEW

UNIX	OS/2	WIN

Change
The default used when WITH CHECK OPTION is specified without keywords has changed from LOCAL in Version 1 to CASCADED in Version 2.

Symptom
This will cause the constraints of all dependant views to be applied.

Resolution
Explicitly specify the LOCAL keyword with the WITH CHECK OPTION to get the same behavior as in Version 1.

SQLSTATE Changes

UNIX	OS/2	WIN

Change
With DB2 Version 2, the SQLSTATEs have been updated to comply with the final published SQL92 standard.

Symptom
In some cases, the value of SQLCA.SQLSTATE will be different than it would be in Version 1 for the same error or situation.

Resolution
If your application is expecting a specific SQLSTATE, you may need to update the value in the comparison.

FOR BIT DATA Comparisons

UNIX	OS/2	WIN

Change
In Version 1, all character strings, including FOR BIT DATA, were compared according to the database collating sequence. In Version 2, character strings with the FOR BIT DATA attribute will be compared according to their bit values, irrespective of the database collating sequence.

Whenever the database manager compares two character strings, if either comparand has the FOR BIT DATA attribute, the comparison is performed with the bit values of the comparands, without consideration of the database collating sequence. If the comparands are of different lengths, there is a logical blank padding (with X'20' on the right) of the shorter string to the length of the longer string.

Symptom
Comparison results will differ from results in Version 1 when the collating sequence and the bit values are in different orders (only for FOR BIT DATA columns). For example, 'A' = x'41' and 'a' = x'61'. 'A' > 'a', however, x'41' < x'61'.

Keep in mind that comparisons take place in many situations including:

- Evaluation of basic predicates
- Use of the ORDER BY clause
- Use of the MIN and MAX column functions

Resolution

You should replicate the data from the FOR BIT DATA column to a column with type CHAR. This will allow the data to be sorted according to the collating sequence instead of their bit values.

Code Page Conversion

UNIX	OS/2	WIN

Change

Code page conversion rules for operands changed in Version 2. These changes improve DB2 compliance with SQL92 standards. It is important to understand that in most cases this will not affect result sets, however, it is possible to find scenarios where output would be different from DB2 Version 1 to Version 2 or to Version 5. In these cases, the output in Version 1 would be the incorrect output from the standpoint of the SQL92 standards compliance.

A few scenarios will be discussed where different output may be experienced:

- When using the LIKE predicate, it will always be the second operand which is converted to the first operand's code page.
- The result type for a UNION ALL set operation is determined in a binary fashion. For queries involving two or more UNION operations, and a mixture of fixed length and varying length character columns, intermediate fixed length datatypes may result in additional trailing blanks. If unequal code pages or columns defined FOR BIT DATA are part this type of operation, the conversion rules are applied to each intermediate result instead of using the final code page throughout the operation.
- The change to consistent conversion rules for result types means that the VALUE scalar function could have a result with a different code page than in previous versions of DB2.

Symptom

The result set may be different since DB2 now adheres to SQL92 standards.

Resolution

There is no resolution as this is an improvement for compliance with SQL92 standards.

Isolation Levels and Blocking All

UNIX	OS/2	WIN

Change
When a cursor is declared without either the FOR UPDATE or FOR READ ONLY clause, it is considered to be an *ambiguous* cursor. If a package containing dynamically declared cursors is bound with the bind option BLOCKING=ALL, but without the bind option LANGLVL=MIA, then any ambiguous cursors will be treated as if FOR READ ONLY had been specified.

Symptom
Your application may receive an SQLCODE of SQL0510N (SQLSTATE 42828) when performing a DELETE WHERE CURRENT OF CURSOR.

Resolution
Rebind with the BLOCKING=UNAMBIG or LANGLVL=MIA options or add a FOR UPDATE clause to the cursor.

ORDER BY Temporary Space Usage

UNIX	OS/2	WIN

Change
Whenever an ORDER BY is performed on a column which does not have an index, a temporary table is used to perform the sort. Beginning in Version 2, LONG VARCHAR and LONG VARGRAPHIC column types will use an increased amount of space as compared to Version 1 in these temporary tables. This may cause the query result rows to exceed the maximum row size (4005 bytes).

Symptom
ORDER BY queries with one or more LONG VARCHAR (or LONG VARGRAPHIC) columns in the SELECT list and for which the select list is physically large, may fail to execute in Version 2 with SQLCODE SQL0670N (SQLSTATE 54010).

Resolution
The following are some ways of attempting to resolve or avoid this scenario:

- Reduce the size of the SELECT list by removing some SELECT items (such as the LONG VARCHAR column(s))
- Apply the SUBSTR function to CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC select items
- Create an index on the ORDER BY fields.

Using Quotes in SQL Statements

UNIX	OS/2	WIN

Change
A defect in previous versions of DB2 allowed double quotes to be used in SQL statements as delimiters of some keywords and operators. For instance, though this is

unpredictable, a query of the form *SELECT C1 "FROM" T1* was processed as if the *FROM* was not delimited.

Beginning in Version 2, this behaviour has been corrected.

Symptom
SQL statements which incorrectly use double quotes to delimit keywords or operators will return errors during statement parsing.

Resolution
The statement syntax should be changed to removed the unnecessary double quotes. For static SQL, if the application source code is unavailable, bind files can be carefully edited to remove the unnessary quotes from the statements. Note that SQL identifiers may require the use of double quotes (these are called delimited identifiers).

Database Security and Tuning

GROUP Authorizations

UNIX	OS/2	

Change
In DB2 Version 1, there was no way to indicate whether a privilege being granted was applicable to a user or to a group. In Version 2, a new field, called GRANTEETYPE, has been added to SYSCAT.DBAUTH, SYSCAT.INDEXAUTH, SYSCAT.PLANAUTH and SYSCAT.TABAUTH. GRANTEETYPE is either a 'U' to represent the GRANTEE is a user or 'G' to represent that the GRANTEE is a group.

During database migration, an attempt is made to determine whether existing privileges defined in the SYSIBM tables are for a user or a group. If the current privileges are for both a user and a group, only the user portion will be represented in the Version 2 database.

Symptom
Loss of authorization if you are a member of a group which is also defined in the operating system as a user.

Resolution
If this access is meant for groups (that is, where the environment variable DB2GROUPS=ON was used in Version 1), then execute the appropriate GRANT command for the appropriate access to the group.

Authentication Type

UNIX	OS/2	WIN

Change
In Version 1, you could provide an authentication type on the CREATE DATABASE command. Beginning in Version 2, this option is ignored. All databases now have the same authentication type as the instance.

Symptom
If the DB2 Version 5 instance authentication type is different than the Version 1 database authentication type, then authentication will behave differently after migration.

Resolution
Make sure that the instance authentication type is the type you want for the databases within that instance.

SYSADM Groups

UNIX	OS/2	

Change
The SYSADM group must be explicitly set in the database manager configuration file.

Symptom
This is automatically taken care of during migration, but a problem could arise if you use a script or command file to change SYSADM groups.

Resolution
Update the script or command file to include the required changes in the database manager configuration file.

Security Enhancements

UNIX	OS/2	

Change
Several security enhancements have been made to the product to make Version 2 and following versions more secure than Version 1. A few of the changes are listed here, however, this is not a complete list.

- Authorization is no longer automatically inferred from file permissions (AIX).
- A general user can execute any DB2 executable, but will not be able to perform the function of that executable unless they have the correct authority. Examples include: `db2start` and `db2trc`. See the *Command Reference* for information on `db2start` and the *Troubleshooting Guide* for information on `db2trc`.
- Authorization for some commands, such as MIGRATE DATABASE, have changed. You should refer to the *Command Reference* and the *API Reference* for the authorization requirements for an individual command or API.

Symptom
You may not be able to execute a DB2 command or API that you used to be able to execute. You will receive a “not authorized” type of SQLCODE.

Resolution
Acquire the proper authorization for the task to be performed.

Utilities and Tools

Executable Name Changes

	OS/2	

Change
The following executables have changed names:

- STARTDBM.EXE is now DB2START.EXE
- STOPDBM.EXE is now DB2STOP.EXE
- SQLPREP.EXE is now the DB2 PREP command
- SQLBIND.EXE is now the DB2 BIND command
- SQLTRC.EXE is now DB2TRC.EXE
- EXPLAIN.EXE is now DB2EXPLN.EXE

Symptom
The original executable names will still be accepted; however, some Version 2 functions are not available (such as new PREP and BIND options).

Resolution
Use the Version 2 executables or commands.

Backup and Restore - BUFF_SIZE Parameter

UNIX	OS/2	

Change
The parameter BUFF_SIZE has changed for the backup and restore APIs. The minimum is now 16 allocation units (of 4K) instead of 8 units, and the increments must be in steps of 16 instead of 1.

Symptom
You may receive a SQLCODE of SQL5130N.

Resolution
Upgrade your application to use a BUFF_SIZE value which is valid for Version 2.

Backup and Restore - Changes Only Option

	OS/2	

Change
In Version 1 there was the ability to backup and restore “Changes Only” to a database. This ability no longer exists. However, applications making the Version 1 API calls will not fail. DB2 simply ignores the second parameter (TYPE) in the sqluback() API call and performs a full backup.

Symptom
A full backup will be taken when specifying a “Changes Only” backup.

Resolution
None exist. “Changes Only” backups are no longer supported.

Backup and Restore - User Exits

	OS/2	

Change
Due to the table space capabilities available beginning in Version 2, it is no longer possible to determine the original location of the backup files. For this reason, user exits which use XCOPY or relied on the database sub-directory format in Version 1 will no longer function beginning in Version 2.

Symptom
If you continue to use User Exits that move the backup files to another location, the restore may not function correctly.

Resolution
User Exits can still be used for log archiving and retrieving. Use the supported parameters and options on the backup command to define the location the backup files will reside.

Backup and Restore - Authority

UNIX	OS/2	

Change
You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the BACKUP command. DBADM authority is no longer sufficient.

Symptom
If you attempt a backup with DBADM authority only, you will be told that you do not have sufficient authority to perform the backup.

Resolution
There are two choices:

1. Log on with an ID that has the proper authority.
2. Set the proper authority for the current ID.

Import - IMPORT REPLACE Option

UNIX	OS/2	WIN

Change
A downlevel client cannot issue an IMPORT REPLACE command to a Version 2 server.

Symptom
If this is attempted, the application will receive an SQL3188N error.

Resolution
There are three possible resolutions to this scenario:

1. Upgrade the client to DB2 Version 5.
2. Execute this command from the DB2 Version 5 server.
3. Split the IMPORT REPLACE into two commands:
   - A DELETE from the table
   - An IMPORT INSERT into the table

REORG - Alternate Path Option

UNIX	OS/2	

Change
The REORG command and API no longer support an “alternate path” as a work area, but rather support the name of a table space to be used as a work area. APIs and commands will not fail, however, this option will be ignored.

Symptom
REORG invocations from downlevel clients will ignore the alternate work path and arbitrarily choose a temporary table space to use as a work area.

Another symptom is you may run out of disk space.

Resolution
Your applications will continue to function, but you should consider upgrading to the DB2 Version 5 calls which contain valid options.

Connectivity and Coexistence

Distributed Transaction Processing - Connect Type

UNIX	OS/2	

Change

In an XA Distributed Transaction Processing environment, such as CICS, applications will always run with connect type 2 as the connection setting. In the last release, connect type 1 was used.

Symptom

It will not be possible to modify the authorization ID on a database connection when the connection already exists.

Resolution

Modification of the authorization ID on a database connection will have to be performed when the connection does not exist.

Distributed Transaction Processing - SQLERRD Changes

UNIX	OS/2	

Change

In an XA Distributed Transaction Processing environment such as CICS, information returned in SQLERRD after a CONNECT has changed. In Version 1, SQLERRD(6) was used to indicate one of the following:

- Non-XA
- DB2/6000 but not supporting XA
- DB2/6000 supporting XA

Beginning in Version 2, SQLERRD(6) is no longer used, but SQLERRD(3) and SQLERRD(4) are used as follows:

SQLERRD(3) Updateablility in the unit of work

- Updateable
- Read Only

SQLERRD(4) Commit type

- One phase commit
- One phase commit, read only
- Two phase commit

Symptom

The sixth SQLERRD element will no longer contain the information wanted by the application.

Resolution
Change the application to look at the third and fourth SQLERRD fields.

DDCS - SQLJSETP

	OS/2	

Change
DDCS for OS/2 used to have a SQLJSETP environment variable. This item had two uses. Each is listed with the DDCS Version 2.3 and DB2 Connect replacement. (In DB2 Version 5, DDCS changed to DB2 Connect.)

1. By placing `/s=e` in the environment variable, bind files containing errors could be bound to DRDA servers. The default is to not allow errors. The `/s=e` function has been replaced by the SQLERROR CONTINUE bind/prep option.

   `/s=c` meant to prep/bind and perform syntax checking only without actually creating a package. This has been replaced by the SQLERROR CHECK bind/prep option.

   `/s=n` meant no errors were allowed during prep/bind. A package would only be created if there were no errors. This has been replaced by the SQLERROR NOPACKAGE prep/bind option.

2. This environment variable also captured prep/bind messages in a file for SQL statements which produced errors. This was needed because when the `/s=e` was specified, all errors were masked and missing from the precompiler generated message file. There is no longer a need for this because all messages are now revealed to the precompiler (and hence its message file) regardless of using SQLERROR CONTINUE or not.

Symptom
The SQLJSETP environment variable and options are ignored, causing prep/bind to work according to their defaults.

Resolution
Use the SQLERROR NOPACKAGE and/or SQLERROR CONTINUE options as needed.

DDCS - DDCSSETP

UNIX		

Change
The DDCSSETP utility has been removed. There is no longer a need for this because all messages are now revealed to the precompiler (and hence its message file) regardless of using SQLERROR CONTINUE or not. Refer to the prep/bind options discussed in “DDCS - SQLJSETP” for information.

DDCS - SQLJTRC.CMD

	OS/2	

Change
DDCS for OS/2 had a utility called `sqljtrc.cmd`. It has been replaced by the `ddcstrc.exe` executable. The invocation syntax has changed.

Symptom
Attempting to trace DB2 Connect will fail if the old utility and parameters are used.

Resolution
Execute the command `ddcstrc.exe` with valid parameters. See the *DB2 Connect User's Guide* for the new syntax.

DDCS - SQLJBIND.CMD

	OS/2	

Change
The DDCS for OS/2 utility called `sqljbind.cmd` has been removed.

Symptom
Attempts to use this utility with DB2 Connect will fail.

Resolution
This utility has been replaced by a three step set of instructions which are described in "Chapter 4, Binding Applications and Utilities" in the *DB2 Connect User's Guide*.

APPC and APPN Nodes

	OS/2	

Change
When using APPC, DB2 for OS/2 Version 1 supported the following commands for cataloging entries in the node directory:

- CATALOG APPN NODE
- CATALOG APPC NODE
- CATALOG NODE

Beginning in Version 2, support for the following has been removed:

- CATALOG APPN NODE
- CATALOG NODE

The CATALOG APPC NODE command has been changed to represent the APPC communication parameters required for the Communications Manager for OS/2 CPI Communications (CPIC) Side Information Profile.

The **symbolic destination name** parameter, in the CATALOG APPC NODE, contains the CPIC Side Information profile name in Communications Manager for OS/2. Refer to the *Command Reference* for details on this command.

Symptom

All of the current connections using APPC will continue to work correctly and DB2 will accept the current catalog information after migration. This is true whether you migrate your server, clients or both.

The current catalogs cannot be modified. If you need to modify the information, you will have to UNCATALOG the node entry, and then recatalog using the new CATALOG APPC command.

If you have existing applications that call the CATALOG NODE API to catalog APPC and APPN node directory entries, these applications will still work. The CATALOG NODE API still supports the Version 1 APPC and APPN API structures.

Resolution

When you try execute the CATALOG APPC node you will need to:

- Create a CPIC Side Information Profile.
- Reference the above profile in the CATALOG APPC NODE command.

Refer to the *DB2 Connect Personal Edition Quick Beginnings* for details on setting these nodes.

Configuration Parameters

ADSM_PASSWORD

OS/2	WIN	UNIX	DB2 PE

Change

In DB2 Version 5, ADSM_PASSWORD is a database configuration parameter. In DB2 Version 2, it was a database manager configuration parameter.

Note: In DB2 PE Version 1, this parameter was spelled `ADSM_PASWD`.

Symptom

Any attempt to update or retrieve the DATABASE MANAGER CONFIGURATION value for ADSM_PASSWORD will be a no-op; that is, no error or value will be returned.

Resolution

You will have to set the ADSM_PASSWORD for any databases for which you want to use the parameter.

MAXDARI and MAXCAGENTS

OS/2	WIN	UNIX	DB2 PE

Change

In Version 2, the value of the MAXDARI and MAXCAGENTS parameters were limited by the value of the MAXAGENTS configuration parameter. The default value of -1 means “equal to MAXAGENTS.”

Beginning in DB2 Version 5, the value of these two parameters are limited by the value of the MAX_COORDAGENTS configuration parameter. The default value of -1 means “equal to MAX_COORDAGENTS.”

Note: On non-Partitioned (non-MPP) configurations, the configuration parameter MAX_COORDAGENTS can only have a value of -1, meaning “equal to MAXAGENTS.”

Symptom

Updates to MAXDARI and MAXCAGENTS to values greater than –1 may fail if the value specified is greater than MAX_COORDAGENTS.

Resolution

Be aware of how MAX_COORDAGENTS is set. MAXDARI and MAXCAGENTS cannot be greater than MAX_COORDAGENTS.

LOGFILSIZ

OS/2	WIN	UNIX	DB2 PE

Change

The data type of this database configuration parameter has changed from being an unsigned 2-byte integer to an unsigned 4-byte integer. A new token has been added for the configuration APIs indicating a 4-byte integer.

For DB2 Version 5, the token is SQLF_DBTN_LOGFIL_SIZ
For DB2 Version 2, the token is SQLF_DBTN_LOGFILSIZ

The configuration API will still recognize the Version 2 token, but the full range of values of this parameter is greater than what is supported by a 2-byte integer.

Symptom

Existing applications will continue to work using the configuration API or via REXX, but the results might be unpredictable because of the larger range in DB2 Version 5.

Resolution

Recode the application or REXX script to use the new token. For users of the Command Line Processor or the Control Center, this change in the token would not affect your applications.

PCKCACHEFILSIZ

OS/2	WIN	UNIX	DB2 PE

Change

The data type of this database configuration parameter has changed from being an unsigned 2-byte integer to an unsigned 4-byte integer. A new token has been added for the configuration APIs indicating a 4-byte integer.

For DB2 Version 5, the token is SQLF_DBTN_PCKCACHE_SIZ
For Version 2, the token is SQLF_DBTN_PCKCACHE_SZ

In Version 2, the value of this parameter was limited by the size of the APPLHEAPSZ configuration parameter and indicated the size of a per-agent parameter. In DB2 Version 5, this parameter limits the size of a global per-database cache. Therefore, its value is no longer limited by the size of the APPLHEAPSZ configuration parameter.

In DB2 PE Version 1.2, the value of this parameter was limited by 7/8 of the value of the DPHEAP configuration parameter, since the cache was allocated from DBHEAP. In DB2 Version 5, the value of the cache is allocated out of its own heap. Therefore, the value of the PCKCACHESZ configuration parameter is no longer limited by the size of the DBHEAP configuration parameter.

Symptom

The following might occur:

- Existing applications will continue to work using the configuration API or via REXX, but the results might be unpredictable because of the larger range in DB2 Version 5.
- Applications might get error code SQL0973, indicating that the PCKCACHE heap has been exhaused.

Resolution

Depending on the symptoms, do one of the following:

- Recode the application to REXX script to use the new token.
- Check the settings of this parameter so that the application reflects the new value.

APPLHEAPSZ and APP_CTL_HEAP_SZ

OS/2	WIN	UNIX	DB2 PE

Change

Beginning in DB2 Version 5, the use of these parameters has changed significantly.

Symptom

Applications might receive an SQL0973 indicating that the APPLHEAP heap or APP_CTL_HEAP has been exhausted.

Resolution

You will have to reconfigure these parameters for optimum performance. Refer to the *Administration Guide* and the online help for the Control Center for recommendations on tuning these parameters.

BUFFPAGE and Multiple Buffer Pools

UNIX	OS/2	WIN	DB2 PE

Change

In previous versions of DB2, each database had one buffer pool, which was created when the database was created. You could change the size of the buffer pool using the *buffpage* parameter. In DB2 Version 5, each database can have multiple buffer pools. You can create additional buffer pools or change the size of a buffer pool through the CREATE BUFFERPOOL or ALTER BUFFERPOOL statements or through the Control Center using the appropriate command.

If the buffer pool size is specified to be –1, then the value of the database configuration parameter is used as the size of the buffer pool.

Note: When the BUFFPAGE database configuration parameter is updated, you will receive an SQLCODE SQL1482W warning.

Symptom

In DB2 Version 5, a new or migrated database has a default buffer pool. For a new database created in DB2 Version 5, the size of the default buffer pool is determined by the operating system. For a migrated database, the size of the buffer pool is set to –1, which then refers to the *buffpage* configuration parameter.

Resolution

To resolve this problem, you will need to do the following:

1. For a new database created in DB2 Version 5, you may change the size of the buffer pool using the ALTER BUFFERPOOL statement.
2. Following the creation or migration of a database, you can then create additional buffer pools for the database using the CREATE BUFFERPOOL statement.

NEWLOGPATH

OS/2	WIN	UNIX	DB2 PE

Change

In DB2 Version 5, in a partitioned database, the node number is appended to the path in the form `path_name \NODExxxx` (`path_name /NODExxxx` on UNIX-based systems), where `xxxx` is the 4 digit node number. This maintains the uniqueness of the path across the database partitions.

Symptom
When updating the NEWLOGPATH configuration parameter, the node number is automatically appended to the path name. This may result in path names that are too long (greater than 242 characters), and the configuration parameter update may fail.

Resolution
Be aware that the log files will reside in the path that includes the node numbering designation. If the configuration parameter update failed, ensure that the path length, including the node number designation, is less than or equal to 242 characters.

MULTIPAGE_ALLOC

			DB2 PE

Change
In DB2 PE Version 1.2, this database configuration parameter was known as MULTIPGAL and the data type of this database configuration parameter was an unsigned 1-byte integer. In DB2 Version 5, the data type of this parameter is an unsigned 2-byte integer, using a new token.

For DB2 Version 5, the token is SQLF_DBTN_MULTIPAGE_ALLOC
For DB2 PE Version 1, the token is SQLF_DBTN_MULTIPGAL

Symptom
Existing applications will continue to work using either the SQLF_DBTN_MULTIPGAL or the SQLF_DBNR_MULTIPAGE_ALLOC tokens.

Resolution
While the configuration APIs support both tokens, applications should be updated to use the new tokens.

EXTENTSIZE vs SEGPAGES

UNIX		

Change
Beginning in Version 2, new *dft_extent_sz* configuration parameter serves as the default EXTENTSIZE setting for table spaces where this is not specified.

- default value: 32 4K pages
- range: 2-256 4K pages

It is modifiable.

Symptom
If an application attempts to specify the SEGPAGES parameter in the CREATE DATABASE command, the command will still work; however, the parameter will be ignored. The EXTENTSIZE will be set to the default.

Resolution
Update the command to specify the new EXTENTSIZE parameter when creating a table space.

LOCKLIST

UNIX	OS/2	

Change
In DB2 Version 5, the size of a lock request block has been changed to 36 bytes. As a result, fewer lock request blocks will fit in the configured amount of space allocated for the lock list.

Symptom
This may result in more frequent lock escalations.

Resolution
You should increase the setting of the LOCKLIST configuration parameter accordingly.

BUFFPAGE and SORTHEAP

UNIX	OS/2	

Change
The tokens for database configuration parameters buffpage and sortheap have changed.

For buffpage:
from SQLF_DBTN_BUFFPAGE to SQLF_DBTN_BUFF_PAGE
For sortheap on OS/2:
from SQLF_DBTN_SORTHEAP to SQLF_DBTN_SORT_HEAP
For sortheap on AIX:
from SQLF_DBTN_SORTHEAPSZ_P to SQLF_DBTN_SORT_HEAP

The names of the parameters as identified in command line processor or in the Control Center remain the same (buffpage and sortheap). The old tokens are maintained for backlevel binary compatibility.

On AIX, the configuration APIs treat the new token and the old token as indicating a 32 byte unsigned integer. On OS/2 however, the configuration APIs will treat the old token as indicating a 16 byte unsigned integer. This is consistent with Version 1 behavior. The new tokens will be treated as indicating an unsigned 32 byte integer.

Symptom
Version 1 applications which specify the old token names will not work against a Version 2 or later database.

Resolution
In order to migrate old application code the token names need to be changed. Additionally, on OS/2, the data type of the variable being passed to the configuration APIs will have to be changed to an unsigned 32 byte integer.

Numeric Values for Database Manager Configuration Tokens

UNIX		

Change
In DB2 for AIX Version 1, the numerical values for the database manager configuration parameter tokens SQLF_KTN_MAXDARI and SQLF_KTN_KEEPDARI were 22 and 23 respectively. Beginning in Version 2, they are 80 and 81 respectively. Binaries from Version 1 will be supported despite this discrepancy.

Symptom
Applications which perform a DATABASE MANAGER CONFIGURATION operation and specify the changed parameters by explicitly stating their numeric values will no longer work as desired.

Resolution
If code is being migrated and the token name is used, nothing needs to be changed. If however, the token values were coded explicitly in the application, the application will have to be changed to reflect the new values.

To protect the application from future changes of this type, it is recommended that the token is coded, rather than the actual value.

Numeric Values for Database Manager Configuration Tokens

	OS/2	

Change
In DB2 for OS/2 Version 1.2, the numerical values for the database manager configuration parameter tokens SQLF_KTN_FILESERVER and SQLF_KTN_OBJECTNAME were 22 and 23 respectively. Beginning in Version 2, they are 47 and 48 respectively. Binaries from Version 1 will be supported despite this discrepancy.

Symptom
Applications which perform a DATABASE MANAGER CONFIGURATION operation and specify the changed parameters by explicitly stating their numeric values will no longer work as desired.

Resolution

If code is being migrated and the token name is used, nothing need be changed. If however, the token values were coded explicitly in the application, the application will have to be changed to reflect the new values.

To protect the application from future changes of this type, it is recommended that the token is coded, rather than the actual value.

New Generic Out-of-Range Return Codes

UNIX	OS/2	WIN

Change

Many return codes indicating an attempt to set a specific parameter outside of its valid range were replaced with generic out-of-range return codes.

The following return codes have been replaced with a return code of -5130 (SQLF_RC_INV_RANGE as defined in sqlutil.h):

- SQLF_RC_INVDB
- SQLF_RC_INVRIO
- SQLF_RC_INVSHPTHR
- SQLF_RC_INVNLL
- SQLF_RC_INVNDBF
- SQLF_RC_INVSCP
- SQLF_RC_INVNAP
- SQLF_RC_INVAHP
- SQLF_RC_INVDHP
- SQLF_RC_INVDLT
- SQLF_RC_INVTAF
- SQLF_RC_INVSHP
- SQLF_RC_INVMAL
- SQLF_RC_INVSTMTHP
- SQLF_RC_INVLOGPRIM
- SQLF_RC_INVLOG2ND
- SQLF_RC_INVLOGFSZ
- SQLF_RC_INVNBP

and SQLF_RC_INV_DBMENT (-5126) is returned, beginning in Version 2, instead of SQLF_RC_INVK3 (-5105) which is no longer returned.

Symptom

If an application is looking for a specific error code which has been replaced by a new one, then this will cause the application to function incorrectly.

Resolution

Update the application to look for valid return codes.

Segments versus 4KB Pages

	OS/2	

Change
All configuration parameters in OS/2 that were expressed in segments in Version 1 are now expressed in 4KB pages.

Symptom
Beginning in Version 2, when you specify a configuration parameter which used to be a measure of segments, it is treated as a measure of 4KB pages. This will result in a different total amount of space in most cases.

Resolution
Migration takes care of this incompatibility by allocating the same amount of storage that was allocated before the migration. Existing applications that specify parameter values should be converted to specify the proper number of 4KB page units.

Obsolete Database Configuration Parameters

	OS/2	

Change
The following database configuration parameters are obsolete:

- AGENTHEAP
- MAXTOTFILOP (there is now a new database manager level configuration parameter by the same name)
- SQLSTMTSZ

Version 1 binary applications attempting to update or get the value of these parameters will result in a no-operation with a return code of 0.

Resolution
Applications should be updated to not reference these parameters.

If you are updating or viewing the value for MAXTOTFILOP, then you can now use Database Manager Configuration commands.

Obsolete Database Manager Configuration Parameters

UNIX	OS/2	

Change
The following database manager configuration parameters are obsolete:

- COMHEAPSZ
- RSHEAPSZ
- SVRIOBLK

- NUMRC
- SQLENSEG (OS/2 only)
- CUINTERVAL

Symptom

Version 1 binary applications attempting to update or get the value of these parameters will result in a no-operation with a return code of 0.

Resolution

Applications should be updated to not reference these parameters.

Appendix C. Memory Usage for DB2 Universal Database Version 5

The following chart presents in summary form the recommended minimum amounts of memory (in megabytes) for each of the editions and features associated with DB2 Universal Database Version 5. The base values include five (5) connections, and in each case, the value given is a liberal amount; the edition or feature will run with less memory, but performance may suffer due to paging.

When multiple editions or features are installed on the same machine (for example, Enterprise and Connect Enterprise), use the highest applicable "Base" as the base amount, and add the "Each Additional" requirement for each edition or feature as required.

Edition/Feature	Platform	Base	Each Additional	Notes
Personal	Intel	32		(1)
Workgroup	Intel	32	0.8	(2), (3)
Enterprise	Intel	32	0.8	(2), (3)
Administration Client	Intel	24		
Workgroup	UNIX	64	1.6	(2), (3)
Enterprise	UNIX	64	1.6	(2), (3)
Extended Enterprise	UNIX	256	2	(4)
Connect Personal	Intel	24		(5)
Connect Enterprise	Intel	64	0.2	
Connect Enterprise	UNIX	64	0.4	
Additional Instance	All		1	(6)
Additional Database	All		2	(6)

Notes:

1. This Base value assumes a "light" application and one small database running on the machine. More than 32MB will be required in some situations.
2. The additional memory values apply to each complex, remote dynamic SQL or CLI application running on the system, above and beyond the base number of connections. Simple, remote static SQL applications require as little as one-eighth the values shown above. Local applications can vary widely in memory requirements, but generally you should allow approximately twenty percent more than the corresponding remote requirement.

3. When intra-partition parallelism is used on the database server, the number of subagents created has a significant effect on memory consumption. If the majority of the activity on the machine makes use of the intra-partition parallelism capabilities, we recommend that the base amount be doubled. The additional amount per connection should be 0.7MB plus 128K per subagent (for Intel); or, 1.3MB plus 256K per subagent (for UNIX). The number of subagents would be the average of all numbers of subagents across all the database server connections.
4. The Base value is per partition. The “Each Additional” value is also per partition, and assumes a typical amount of parallelism activated by DB2 for each connection within that partition.
5. 24MB is recommended for Windows NT. 16MB is normally sufficient for Windows 95 and OS/2. These figures assume a “light” application running on the machine; it is not unusual for some applications to require 32MB.
6. The additional memory for each instance or database assumes that the default configuration parameters are in use. Settings of parameters such as BUFFPAGE can consume very large amounts of additional memory.

Regarding the amount of memory required for DB2 clients: The maximum size is approximately 2MB. However, this size could be significantly less depending on your platform; or more, depending on the application overhead.

Appendix D. Naming Rules

Use the naming rules shown below when you provide names for the following databases and database objects:

- Database Names
- Database and Database Alias Names
- User IDs and Passwords
- Schema Names
- Group and User Names
- Object Names

Do not use IBM SQL or ISO/ANSI SQL92 reserved words to name tables, views, columns, indexes, or authorization IDs. A list of these words is included in the *SQL Reference* manual.

Refer to the *Quick Beginnings* manuals for naming rules about authorization IDs (including user names and group names) and workstations, and for additional platform restrictions.

Database Names

Every time a new database is created, the database manager creates a separate directory to store the control files and data files for that database.

The naming scheme for these directories is `SQL00001` through `SQLnnnnn`, where `SQL00001` contains control files associated with the first database created, `SQL00002` contains control files for the second database created, and so on.

These directories are maintained automatically. To avoid potential directory naming problems, do not create your own directories using the same naming schema as used by the database manager, and do not manipulate directories that have already been created by the database manager.

Database and Database Alias Names

Database names are the identifying names you or your users provide as part of the CREATE DATABASE command or API. These names must be unique within the location in which they are cataloged. For example, for UNIX-based implementations of DB2, this location is a directory path, while in OS/2 implementations it is a drive letter.

Database alias names are local synonyms given to local or remote databases. These names must be unique within the System Database Directory, in which all aliases are stored for the individual instance of the database manager. When a new database is created, the alias defaults to the database name. As a result, you cannot create a database using a name that exists as a database alias, even if there is no database with that name.

When naming a database or a database alias, the name you specify:

- Can contain 1 to 8 characters
- Must begin with one of the following:
  - A through Z (converts lowercase letters to uppercase)
  - @, #, or $
- Other characters can include:
  - A through Z (converts lowercase letters to uppercase)
  - 0 through 9
  - @, #, $, and _ (underscore)

Note: To avoid potential problems, do not use the special characters @, #, and $ in a database name if you intend to use the database in a communications environment. Also, because these characters are not common to all keyboards, do not use them if you plan to use the database in another country. Finally, on Windows NT systems, ensure that no instance name is the same as a service name.

User IDs and Passwords

When creating a user ID or password, the name you create:

- Cannot be any of the following:
  - USERS, ADMINS, GUESTS, PUBLIC, LOCAL, or any SQL reserved word listed in the *SQL Reference* manual.
- Cannot begin with:
  - SQL, SYS, or IBM
- Other characters can include:
  - A through Z

    Note: Some operating systems allow case-sensitive user IDs and passwords. You should check with your operating system information to see if this is the case.
  - 0 through 9
  - @, #, or $

Schema Names

The following schema names are reserved words and must not be used:

- SYSCAT
- SYSFUN
- SYSIBM
- SYSSTAT

In general, you should avoid schema names that begin with SYS to avoid potential migration problems in the future. The database manager will not allow you to create triggers, user-defined types or user-defined functions using a schema name beginning with SYS.

Group and User Names

On UNIX, groups and users can have the same name. For the GRANT statement you must specify whether you are referring to a group or a user. For the REVOKE statement specifying user or group depends on whether or not there are multiple rows in the authorization catalog tables for the GRANTEE with different values of GRANTEETYPE.

On OS/2, groups and users cannot have the same name.

On Windows NT, Local Group names, Global Group names and User IDs cannot have the same name.

Object Names

Database objects include the following:

- Schemas
- Tables
- Views
- Columns
- Indexes
- User-defined functions (UDFs)
- User-defined types (UDTs)
- Triggers
- Aliases
- Table spaces
- Stored procedures
- Nodegroups
- Buffer pools
- Event monitors

When naming database objects, the name you specify:

- Can contain 1 to 18 characters (bytes)

  Note: Schemas are an exception: They can only allow 1 to 8 characters.

- Must begin with one of the following:
  - A through Z (converts lowercase letters to uppercase)
  - A valid accented letter (such as ö)
  - A multibyte character, except multibyte spaces (for multibyte environments)
- Other characters can include:
  - A through Z (converts lowercase letters to uppercase)
  - A valid accented letter (such as ö)
  - 0 through 9
  - @, #, $, and _ (underscore)
  - Multibyte characters, except multibyte spaces (for multibyte environments)

- Keywords can be used. If the keyword is used in a context where it could also be interpreted as an SQL keyword, it must be specified as a delimited identifier. Refer to the *SQL Reference* for information on delimited identifiers.
- For maximum portability, use the IBM SQL and ISO/ANSI SQL92 reserved words. For a list of these words, refer to the *SQL Reference* manual.

Notes:

1. Using delimited identifiers, it is possible to create an object that violates these naming rules; however, subsequent use could lead to error situations. To avoid potential problems with the use and operation of your database, **do not** violate the above rules.

   For example, if you created a column with a + or – sign included in the name and you subsequently use that column in an index, you will experience problems when you attempt to reorganize the table.
2. For information about National Language Support (NLS) related to object names, see Appendix M, “National Language Support (NLS)” on page 813.

Appendix E. DB2 Registry Values and Environment Variables

The following is a brief list of DB2 registry values and environment variables that you may need to know about to get up and running. Each has a brief description; some may not apply to your environment.

Parameter	Operating System	Values	Description
		General	
DB2ACCOUNT	All	Default=(not set)	The accounting string that is sent to the remote host. Refer to the *DB2 Connect User's Guide* for details.
DB2CODEPAGE	All	Default: derived from the language ID, as specified by the operating system.	Specifies the code page of the data presented to DB2 for database client application. The user should not set *db2codepage* unless explicitly stated in DB2 documents, or asked to do so by DB2 service. Setting *db2codepage* to a value not supported by the operating system can produce unexpected results. Normally, you do not need to set *db2codepage* because DB2 automatically derives the code page information from the operating system.
DB2COUNTRY	All	Default: derived from the language ID, as specified by the operating system.	Specifies the country code of the client application, which influences date and time formats.
DB2DBDFT	All	Default=(not set)	Specifies the database alias name of the database that will be implicitly connected to when applications are started and no implicit connect has been done. This keyword is ignored if it is set.
DB2DBMSADDR	Windows 95	Default=(not set), Value: (90000000)	Sets the memory location from which the Windows 95 system allocates shared memory for an instance. For each additional instance you must specify a different memory location from within the range 0x80000000 to 0xBFFFFFFF. On a Windows 95 system, you need to override the default value (90000000) only if you get an error message indicating you could not create database shared memory.

Parameter	Operating System	Values	Description
DB2DISCOVERYTIME	OS/2, Windows 95, and Windows NT	Default=40 seconds, Minimum=20 seconds	Specifies the amount of time that SEARCH discovery will search for DB2 systems.
DB2INCLUDE	All	Default=current directory	Specifies a path to be used during the processing of the SQL INCLUDE text-file statement during DB2 PREP processing. It provides a list of directories where the INCLUDE file might be found. Refer to the *Embedded SQL Programming Guide* for descriptions of how *db2include* is used in the different precompiled languages.
DB2INSTDEF	OS/2, Windows 95 and Windows NT	Default=DB2	Sets the value to be used if *DB2INSTANCE* is not defined.
DB2OPTIONS	All except Windows 3.1 and Macintosh	Default=null	Sets command line processor options.
DB2SLOGON	Windows 3.1	Default=NO	Enables a secure logon in DB2 for Windows 3.1. If *db2slogon*=YES DB2 does not write user IDs and passwords to a file, but instead uses a segment of memory to maintain them. When *db2slogon* is enabled, the user must logon each time Windows 3.1 is started.
DB2TIMEOUT	Windows 3.1 and Macintosh	Default=(not set)	Used to control the timeout period for Windows 3.1 and Macintosh clients during long SQL queries. After the timeout period has expired a dialog box pops up asking if the query should be interrupted or allowed to continue. The minimum value for this variable is 30 seconds. If *db2timeout* is set to a value between 1 and 30, the default minimum value will be used. If *db2timeout* is set to a value of 0, or a negative value, the timeout feature is disabled. This feature is disabled by default.

Parameter	Operating System	Values	Description
DB2TRACENAME	Windows 3.1 and Macintosh	Default= DB2WIN.TRC	On Windows 3.1 and Macintosh, specifies the name of the file where trace information is stored. The default is *db2tracename*=DB2WIN.TRC, and is saved in your current instance directory (for example, \sqllib\db2). We strongly recommend that you specify the full path name when naming the trace file.
DB2TRACEON	Windows 3.1 and Macintosh	Default=N, Values: Y, N	On Windows 3.1 and Macintosh, turns trace on to provide information to IBM in case of a problem. (It is not recommended that you turn trace on unless you encounter a problem you cannot resolve.) Refer to the *Troubleshooting Guide* for information on using the trace facility with DB2 Client Application Enabler.
DB2TRCFLUSH	Windows 3.1 and Macintosh	Default=N, Values: Y, N	On Windows 3.1 and Macintosh, *db2trcflush* can be used in conjunction with *db2traceon*=Y. Setting *db2trcflush*=Y will cause each trace record to be written immediately into the trace file. This will slow down your DB2 system considerably, so the default setting is *db2trcflush*=N. This setting is useful in cases where an application hangs the system and requires the system to be rebooted. Setting this keyword guarantees that the trace file and trace entries are not lost by the reboot.
DB2TRCSYSERR		Default=1, values: 1 - 32767	Specifies the number of system errors to trace before the client turns off tracing. The default value traces one system error, after which, trace is turned off.

Parameter	Operating System	Values	Description
DB2YIELD	Windows 3.1	Default=N, Values: Y, N	Specifies the behavior of the Windows 3.1 client while communicating with a remote server. When `N` is set, the client will not yield the CPU to other Windows 3.1 applications, and the Windows environment is halted while the client application is communicating with the remote server. You must wait for the communications operation to complete before you can resume any other tasks. When set to `Y`, your system functions as normal. It is recommended that you try to run your application with *db2yield*=`YES`. If your system crashes, you will need to set *db2yield*=`NO`. For application development, ensure your application is written to accept and handle Windows messages while waiting for a communications operation to complete.
System Environment			
DB2INSTANCE	All	Default=*db2instdef* on OS/2, Windows 95, and Windows NT.	The environment variable used to specify the instance that is active by default. On UNIX, users must specify a value for *DB2INSTANCE*.
DB2INSTPROF	OS/2, Windows 95, and Windows NT		The environment variable used to specify the location of the instance directory on OS/2, Windows 95 and Windows NT operating systems, if different than *DB2PATH*.
DB2PATH	OS/2, Windows 95, and Windows NT		The environment variable used to specify the directory where the product is installed on OS/2, Windows 95 and Windows NT operating systems. By default it is set to x:\sqllib\win on Windows 95 and Windows NT.
Communications			
DB2COMM	All, server only	Default= null, Values: any combination of APPC,IPXSPX, NETBIOS,NPIPE, TCPIP	Specifies the communication managers that are started when the database manager is started. If this is not set, no DB2 communications managers are started at the server.

Parameter	Operating System	Values	Description
DB2NBADAPTERS	OS/2 and Windows NT, server only	Default=0, Range: 0-15, Multiple values should be separated by commas	Used to specify which local adapters to use for DB2 NetBIOS LAN communications. Each local adapter is specified using its logical adapter number.
DB2NBCHECKUPTIME	OS/2 and Windows NT, server only	Default=1 minute, Values: 1-720	Specifies the time interval between each invocation of the NetBIOS protocol checkup procedure. Checkup time is specified in minutes. Lower values will ensure that the NetBIOS protocol checkup runs more often, freeing up memory and other system resources left when unexpected agent/session termination occurs.
DB2NBINTRLISTENS	OS/2 and Windows NT, server only	Default=1, Values: 1-10 Multiple values should be separated by commas	Specifies the number of NetBIOS listen send commands (NCBs) that will be asynchronously issued in readiness for remote client interrupts. This flexibility is provided for "interrupt active" environments to ensure that interrupt calls from remote clients will be able to establish connections when servers are busy servicing other remote interrupts. Setting *db2nbintrlistens* to a lower value will conserve NetBIOS sessions and NCBs at the server. However, in an environment where client interrupts are common, you may need to set *db2nbintrlistens* to a higher value in order to be responsive to interrupting clients. **Note:** Values specified are position sensitive; they relate to the corresponding value positions for *db2nbadapters*.
DB2NBBRECVBUFFSIZE	OS/2 and Windows NT, server only	Default=4096 bytes, Range: 4096-65536	Specifies the size of the DB2 NetBIOS protocol receive buffers. These buffers are assigned to the NetBIOS receive NCBs. Lower values conserve server memory, while higher values may be required when client data transfers are larger.

Parameter	Operating System	Values	Description
DB2NBBRECVNCBS	OS/2 and Windows NT, server only	Default=10, Range: 1-99	Specifies the number of NetBIOS "receive_any" commands (NCBs) that the server will issue and maintain during operation. This value may be adjusted depending on the number of remote clients to which your server is connected. Lower values will conserve server resources. **Note:** Each adapter in use can have its own unique receive NCB value specified by *db2nbbrecvncbs*. The values specified are position sensitive; they relate to the corresponding value positions for *db2nbadapters*.
DB2NBRESOURCES	OS/2, server only	Default=(not set)	Specifies the number of NetBIOS resources to allocate for DB2 use in a multi-context environment. This variable is restricted to multi-context client operation.
DB2NBSENDNCBS	OS/2 and Windows NT, server only	Default=6, Range: 1-720	Specifies the number of send NetBIOS commands (NCBs) that the server will reserve for use. This value may be adjusted depending on the number of remote clients your server is connected to. Setting *db2nbsendncbs* to a lower value will conserve server resources. However, you may need to set it to a higher value to prevent the server from waiting to send to a remote client when all other send commands are in use.
DB2NBSESSIONS	OS/2 and Windows NT, server only	Range: 5-254	Specifies the number of sessions that DB2 should request to be reserved for DB2 use. The value of *db2nbsessions* can be set to request a specific session for each adapter specified using *db2nbadapters*. **Note:** Values specified are position sensitive; they relate to the corresponding value positions for *db2nbadapters*.

Parameter	Operating System	Values	Description
DB2NBXTRANCBS	OS/2 and Windows NT, server only	Default=5 per adapter, Range: 5-254	Specifies the number of "extra" NetBIOS commands (NCBs) the server will need to reserve when the **db2start** command is issued. The value of *db2nbxtrancbs* can be set to request a specific session for each adapter specified using *db2nbadapters*.
DB2NETREQ	Windows 3.1	Default=3, Range: 0-25	Specifies the number of NetBIOS requests that can be run concurrently on Windows 3.1 clients. The higher you set this value, the more memory below the 1MB level will be used. When the concurrent number of requests to use NetBIOS services reaches the number you have set, subsequent incoming requests for NetBIOS services are held in a queue and become active as the current requests complete. If you enter 0 (zero) for *db2netreq*, the Windows database client issues NetBIOS calls in synchronous mode using the NetBIOS wait option. In this mode, the database client allows only the current NetBIOS request to be active and does not process another one until the current request has completed. This can affect other application programs. The 0 value is provided for backwards compatibility only. It is strongly recommended that 0 not be used.
DB2SERVICETPINSTANCE	OS/2 and Windows NT	Default=(not set)	Used to support incoming APPC connections from DB2 workstation V.1 clients or from the DB2 MVS database. When the **db2start** command is invoked, the instance specified will start the APPC listeners for the following TP names: DB2INTERRUPT x'07'68 x'07'6SN
DCE Directories			

Parameter	Operating System	Values	Description
DB2DIRPATHNAME	OS/2 and UNIX		Specifies a temporary override of the DIR_PATH_NAME parameter value in the database manager configuration file. If a directory server is used and the target of a CONNECT statement or ATTACH command is not explicitly cataloged, then the target is concatenated with DB2DIRPATHNAME (if specified) to form the fully qualified DCE name. **Note:** The *db2dirpathname* value has no effect on the instance's global name, which is always identified by the database manager configuration parameters DIR_PATH_NAME and DIR_OBJ_NAME.
DB2CLIENTCOMM	OS/2 and UNIX		Specifies a temporary override of the DFT_CLIENT_COMM parameter value in the database manager configuration file. If both DFT_CLIENT_COMM and *db2clientcomm* are not specified, then the first protocol found in the object is used. If either one or both of them are specified, then only the first matching protocol will be used. In either case, no retry is attempted if the first connect fails.
DB2CLIENTADPT	OS/2	Default=null, Range: 0-15	Specifies the client adapter number for NETBIOS protocol on OS/2 operating systems. The *db2clientadpt* value overrides the DFT_CLIENT_ADPT parameter value in the database manager configuration file.
DB2ROUTE	OS/2 and UNIX		Specifies the name of the Routing Information Object the client uses when it connects to a database with a different database protocol. The *db2route* value overrides the ROUTE_OBJ_NAME parameter value in the database manager configuration file.
Command Line Processor			

Parameter	Operating System	Values	Description
DB2BQTIME		Default=1 second, Maximum value: 1 second.	Specifies the amount of time the command line processor front end will sleep before checking if the back end process is active and establishing a connection to it.
DB2BQTRY		Default=60 retries, Minimum value: 0retries.	Specifies the number of times the command line processor front end process tries to determine whether the back end process is already active. It works in conjunction with *db2bqtime*.
DB2IQTIME		Default=5 seconds, Minimum value: 1 second.	Specifies the amount of time the command line processor back end process waits on the input queue for the front end process to pass commands.
DB2RQTIME		Default=5 seconds, Minimum value: 1 second.	Specifies the amount of time the command line processor back end process waits for a request from the front end process.
MPP Configuration			
DB2NODE	All	Default=(not set) Values: 1-999	Specifies which node of the MPP server instance you want to connect or attach to.
DB2_SELECT_TIMEOUT_INIT	AIX	Default=1000 microseconds, Range: 10-999990	Specifies the initial wait time in microseconds that Fast Communication Manager (FCM) waits to examine the local and remote work queues to detect messages. When there are no messages on a given pass, timeout starts. When a message is detected by FCM the timeout value is immediately reset to *db2_select_timeout_init*. **Note:** Use *db2_select_timeout_init* in conjunction with *db2_select_timeout_incr* and *db2_select_timeout_max* to control the timeout period for the FCM daemon's priority and responsiveness in an MPP configuration. The value specified for *db2_select_timeout_init* must not exceed the value specified for *db2_select_timeout_max*.

Parameter	Operating System	Values	Description
DB2_SELECT_TIMEOUT_INCR	AIX	Default=10, Range: 10-999990	Specifies the increment on wait time between passes each time that Fast Communication Manager (FCM) examines the local and remote work queues and finds no messages. **Note:** Use *db2_select_timeout_incr* in conjunction with *db2_select_timeout_init* and *db2_select_timeout_max* to control the FCM daemon's priority and responsiveness in an MPP configuration.
DB2_SELECT_TIMEOUT_MAX	AIX	Default=30000 microseconds, Range: 10-999990	Specifies the maximum time between passes that Fast Communication Manager (FCM) will wait to examine the local and remote work queues. **Note:** Used in conjunction with *db2_select_timeout_init* and *db2_select_timeout_incr* to control the FCM daemon's priority and responsiveness in an MPP configuration.
Miscellaneous			
DB2ADMINSERVER	OS/2, Windows 95 and Windows NT, and UNIX		Specifies which DB2 instance is set up as the DB2 Administration Server.
DB2AUTOSTART	UNIX, server only	Default=(not set), Value: YES	Specifies whether the instance will be auto-started on the system reboot or not. To auto-start an instance, set this environment variable to `YES`. To disable auto-starting an instance, unset this environment variable. By default, *db2autostart* is not set.
DB2_AVOID_PREFETCH	All	Default=OFF	Specifies whether or not prefetch should be used during crash recovery. If *db2_avoid_prefetch*=`ON`, prefetch is not used.
DB2CHKPTR		Default: OFF, Values: ON or OFF	Specifies whether or not pointer checking for input is required.

Parameter	Operating System	Values	Description
DB2CLIINIPATH			Used to override the default path of the DB2 CLI/ODBC configuration file (db2cli.ini) and specify a different location on the client. The value specified must be a valid path on the client system.
DB2DEFPREP	All	Default=NO, Values: ALL, YES, or NO	Simulates the runtime behavior of the DEFERRED_PREPARE precompile option for applications that were precompiled prior to this option becoming available. For example, if a DB2 v2.1.1 or earlier application were run in a DB2 v2.1.2 or later environment, *db2defprep* could be used to indicate the desired 'deferred prepare' behavior.
DB2DMNBCKCTLR	NT	Default=null	If DB2 is installed on a backup domain controller, setting *db2dmnbckctlr*=YES allows DB2 to use the security database on the backup domain controller, thereby reducing LAN traffic. **Note:** A backup domain controller shadows the security database on the primary domain controller.
DB2_GRP_LOOKUP	NT	Default=null	Specifies which NT security mechanism will be used to enumerate the groups that a user belongs to.
DB2_INDEX_FREE		Default=10, Range: 0-60	Specifies the amount of free space left on all index pages during CREATE INDEX (both SQL and LOAD).
DB2LOADREC			Used to override the location of the load copy during roll forward. If the user has changed the physical location of the load copy, *db2loadrec* must be set before issuing the roll forward.

Parameter	Operating System	Values	Description
DB2LOCK_TO_RB			Specifies whether lock timeouts cause the entire transaction to be rolled-back, or only the current statement. If *db2lock_to_rb* is set to STATEMENT before the **db2start** command is issued, locked timeouts cause only the current statement is rolled back. Any other setting results in transaction rollback.
DB2_MMAP_READ	AIX	Default=ON, Values: ON, OFF	Used in conjunction with *db2_mmap_write* to allow DB2 to use mmap as an alternate method of i/o. In most environments, mmap should be used to avoid operating system locks when multiple processes are writing to different sections of the same file. However, if you do not have a parallel i/o problem, you may set *db2_mmap_read* and *db2_mmap_write* to OFF to free up the AIX memory segment.
DB2_MMAP_WRITE	AIX	Default=ON, Values: ON, OFF	Used in conjunction with *db2_mmap_read* to allow DB2 to use mmap as an alternate method of i/o. In most environments, mmap should be used to avoid operating system locks when multiple processes are writing to different sections of the same file. However, if you do not have a parallel i/o problem, you may set *db2_mmap_read* and *db2_mmap_write* to OFF to free up the AIX memory segment.
DB2NTNOCACHE	Windows NT	Default=(not set), Value: If present, must be set to 1	Specifies whether or not DB2 will open database files with a NOCACHE option. If *db2ntnocache*=1, file system caching is eliminated. If *db2ntnocache* is not set, the operating system caches DB2 files. This applies to all data except for files that contain LONG FIELDS or LOBS. Eliminating system caching allows more memory to be available to the database so that the bufferpool or sortheap can be increased.

Parameter	Operating System	Values	Description
DB2NTREMOTEPREG	Windows 95 and Windows NT	Default=(not set) Value: Any valid Windows 95 or Windows NT machine name	Specifies the remote machine name that contains the Win32 registry list of DB2 instance profiles and DB2 instances. The value for *db2remotepreg* should only be set once after DB2 is installed, and should not be modified. Use this variable with extreme caution.
DB2NTWORKSET	Windows NT	Default=1,1	Used to modify the minimum and maximum working set size available to DB2. When you are not in a paging situation, a process's working set can grow as large as needed. When paging occurs, the maximum working set that a process can have is approximately 1 MB. Specify *db2ntworkset* for DB2 using the syntax *db2ntworkset*=`min,max`, where min and max are expressed in megabytes.
DB2PRIORITIES			Controls the priorities of DB2 processes and threads. The value for *db2priorities* must be set before the **db2start** command is issued.
DB2RAWLOGS	UNIX	Default=OFF	Allows the use of raw disk devices such as logs. When using raw database logs you must set *db2rawlogs*=`TRUE` before issuing the **db2start** command.
DB2_RR_TO_RS		Default=NO, Values: YES, NO	Specifies whether or not next-key locking occurs on access to user tables. When set to YES, next-key locking will not be activated.
DB2SORCVBUF	Windows 95 and Windows NT	Default=32767	Specifies the value of TCP/IP receive buffers on Windows 95 and Windows NT operating systems.
DB2SORT	All, server only	Default=null	Specifies the location of a library to be loaded at runtime by the load utility. The library contains the entry point for functions used in sorting indexing data. Use *db2sort* to exploit vendor-supplied sorting products for use with the LOAD utility in generating table indexes. The path supplied must be relative to the database server.

Parameter	Operating System	Values	Description
DB2SOSNDBUF	Windows 95 and Windows NT	Default=32767	Specifies the value of TCP/IP send buffers on Windows 95 and Windows NT operating systems.
DB2SYSTEM	Windows NT, OS/2, UNIX		Specifies the name that is used by your users and database administrators to identify the DB2 server system. If possible, this name should be unique within your network. This name is displayed in the system level of the Control Center's object tree to aid administrators in the identification of server systems that can be administered from the Control Center. When using the 'Search the Network' function of the Client Configuration Assistant, DB2 discovery returns this name and it is displayed at the system level in the resulting object tree. This name aids users in identifying the system that contains the database they wish to access. A value for *db2system* is set at installation time as follows: On Windows NT, the setup program sets it equal to the computer name specified for the Windows NT system. On OS/2, the user is prompted to enter the DB2SYSTEM name during the installation process. On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.
DB2THREADIF			Disables DB2's support for multi-threaded applications. We recommend that *db2threadif* not be used, except under the the direction of IBM service.
DB2UPMPR	OS/2	Default=ON, Values: ON or OFF	Specifies whether or not the UPM logon screen will display onscreen when the user enters the wrong userid or password on OS/2.

Appendix F. Using Distributed Computing Environment (DCE) Directory Services

DCE provides the Cell Directory Service (CDS) and Global Directory Service (GDS). For more information about DCE concepts and these services, refer to the *Introduction to OSF DCE* manual. The DB2 function for DCE Directory Services supports CDS only. With this support, the user does not have to create each database, node, and DCS database on every single client. All of this information is centralized in DCE CDS.

The following sections describe how to setup and access a database using DCE Directory Services:

- Creating Directory Objects
- Attributes of Each Object Class
- Directory Services Security
- Configuration Parameters and Environment Variables
- CATALOG, CONNECT, and ATTACH Commands
- How a Client Connects to a Database
- How Directories are Searched
- Temporarily Overriding DCE Directory Information
- Directory Services Tasks
- Directory Services Restrictions

DCE directory services may not be supported by all DB2 clients. If DCE directory services is supported for a DB2 client, your *Quick Beginnings* manual provides additional information.

Creating Directory Objects

There are three types of directory objects that the database administrator needs to create:

- "Database Objects"
- "Database Locator Objects" on page 649
- "Routing Information Objects" on page 650

Each object contains attributes. Refer to "Attributes of Each Object Class" on page 651 for a complete description of the attributes.

Before the database administrator can create the objects, the DCE administrator needs to add database information into a CDS table and grant create privileges to the database administrator. Refer to "DCE Administrator Tasks" on page 666 for the details.

Database Objects

A database object is required for each target database. The object has a name that contains the cell name concatenated to the directory name and the name of the database, for example:

```
/.../cell_name/dir_name1/dir_name2/OBJ_NAME
```

Note: The following is recommended for the name of the database. The name should be less than or equal to 8 characters and all the characters should be upper case. If the name is mixed case or longer than 8 characters, you need to use the CATALOG GLOBAL DATABASE command to assign an alias. See "CATALOG GLOBAL DATABASE Command" on page 659 for details about the command.

The following is an example of a database object. The object stored in the DCE directory contains other information such as a timestamp. The letter to the left of each attribute indicates if the attribute is required - R, optional - O, or a comment - C.

```
   Object name:              /.../CELL_TORONTO/subsys/database/AIXDB1
R  DB_Object_Type:           D
C  DB_Product_Name:          DB2_for_AIX
C  DB_Product_Release:       V5R1M000
R  DB_Native_Database_Name:  AIXDBASE
R  DB_Database_Protocol:     DB2RA
R  DB_Authentication:        CLIENT
O  DB_Communication_Protocol:
O  DB_Database_Locator_Name: /.../CELL_TORONTO/subsys/database/AIX_INST
C  DB_Comment:               Test_database_on_AIX
```

If the database is one of many databases associated with a database manager instance, the database object should contain the name of a database locator object and the communication protocol should be blank. The name of the database locator object is the fully-qualified name of the database manager or DB2 Connect instance.

Here is an example of the DCE commands to create the object. Before any objects can be created, the DCE administrator needs to do the steps described in "DCE Administrator Tasks" on page 666.

First you must type the following in a file called *cdscp.inp*:

```
create object /.:/subsys/database/AIXDB1

add object /.:/subsys/database/AIXDB1 DB_Object_Type            = D
add object /.:/subsys/database/AIXDB1 DB_Product_Name           = DB2_for_AIX
add object /.:/subsys/database/AIXDB1 DB_Product_Release        = V5R1M000
add object /.:/subsys/database/AIXDB1 DB_Native_Database_Name   = AIXDBASE
add object /.:/subsys/database/AIXDB1 DB_Database_Protocol      = DB2RA
add object /.:/subsys/database/AIXDB1 DB_Authentication         = CLIENT
add object /.:/subsys/database/AIXDB1 DB_Database_Locator_Name  = /.../CELL_TORONTO/subsys/database/AIX_INST
add object /.:/subsys/database/AIXDB1 DB_Comment                = Test_database_on_AIX
```

Then you must run either

- dcelogin principal password (on OS/2); or,
- dce_login principal password (on UNIX).

This should be followed by

- cdscp < cdscp.inp

Use the following command to display the object:

```
cdscp show object /.:/subsys/database/AIXDB1
```

If the database is the only database associated with a database manager instance, the database object should contain values for the Communication Protocol attribute and the name of the database locator object should be blank. For example:

```
   Object name:                /.../CELL_TORONTO/subsys/database/MVSDB
R  DB_Object type:             D
C  DB_Product_Name:            DB2_for_MVS
C  DB_Product_Release:         V5R1M00
R  DB_Native_Database_Name:    MVSDBASE
R  DB_Database_Protocol:       DRDA
R  DB_Authentication:          SERVER
O  DB_Communication_Protocol:  APPC;NET1;TARGETLU1;DB2DRDA;MODE1;PROGRAM
O  DB_Database_Locator_Name:
C  DB_Comment:                 Test_database_on_MVS
```

Database Locator Objects

These objects contain the details about all the communication protocols used by a DBMS instance or a DB2 Connect instance. One database locator object is required for:

- Each instance with both DBMS and DB2 Connect
- Each DBMS instance which is associated with more than one database, but without an associated DB2 Connect
- Each DB2 Connect instance without an associated DBMS.

The object has a name that contains the cell name concatenated to the directory name and the one-part name of the database instance, for example:

```
/.../cell_name/dir_name1/dir_name2/AIX_INST
```

Note: If the instance is used as the target of an ATTACH, the one-part name must be less than or equal to 8 characters and all upper case.

The following is an example of a database locator object. The object stored in the DCE directory contains other information such as a timestamp. The letter to the left of each attribute indicates if the attribute is required - R, optional - O, or a comment - C.

```
   Object name:                /.../CELL_TORONTO/subsys/database/AIX_INST
R  DB_Object_Type:             L
C  DB_Product_Name:            DB2_for_AIX
C  DB_Product_Release:         V5R1M00
R  DB_Communication_Protocol: TCPIP;HOSTNAME1;1234
R  DB_Communication_Protocol: APPC;NET1;TARGETLU1;TPN1;MODE;PROGRAM
C  DB_Comment:                 Test_instance_on_AIX
```

When an attribute is defined in both the database object and the database locator object, the value in the database object is used.

Here is an example of the DCE commands to create the object. Before any objects can be created, the DCE administrator needs to do the steps described in “DCE Administrator Tasks” on page 666.

First you must type the following in a file called *cdscp.inp*:

```
create object /.:/subsys/database/AIX_INST

add object /.:/subsys/database/AIX_INST DB_Object_Type            = L
add object /.:/subsys/database/AIX_INST DB_Product_Name           = DB2_for_AIX
add object /.:/subsys/database/AIX_INST DB_Product_Release        = V5R1M00
add object /.:/subsys/database/AIX_INST DB_Communication_Protocol = TCPIP;HOSTNAME1;1234
add object /.:/subsys/database/AIX_INST DB_Communication_Protocol = APPC;NET1;TARGETLU;TPN1;MODE;PROGRAM
add object /.:/subsys/database/AIX_INST DB_Comment                = Test_instance_on_AIX
```

Then you must run either

- dcelogin principal password (on OS/2); or,
- dce_login principal password (on UNIX).

This should be followed by

- cdscp < cdscp.inp

Use the following command to display the object:

```
cdscp show object /.:/subsys/database/AIX_INST
```

Routing Information Objects

Routing information objects are required for host access. When a mismatch exists in the database protocol used by a client and the database protocol used by the target database, the routing object tells the client which DB2 Connect instance to use. Attributes exist for each target database, which include the database protocols that are available and the name of the database locator object for the DB2 Connect instance. The object has a name that contains the cell name concatenated to the directory name and a unique one-part name, for example:

```
/.../cell_name/dir_name1/dir_name2/ROUTE1
```

The following is an example of a routing information object. The object stored in the DCE directory contains other information such as a timestamp. The letter to the left of each attribute indicates if the attribute, and each token within an attribute is required - R, optional - O, or a comment - C.

Client group 1 is Client_1, Client_2, and Client_3 in Figure 63 on page 660.

```
   Object name:    /.../CELL_TORONTO/subsys/database/ROUTE1
R  DB_Object_Type: R
C  DB_Comment:     Routing_for_client_group_1

R  DB_Target_Database_Info
   R  Database name                 = /.../CELL_TORONTO/subsys/database/MVSDB
   R  Outbound protocol from router = DRDA
   R  Inbound protocol to router    = DB2RA
   R  Authenticate at gateway       = 1
   O  Parameter string              = NOMAP,D,INTERRUPT_ENABLED
   R  DB_Database_Locator_Name      = /.../CELL_TORONTO/subsys/database/GW_INST

R  DB_Target_Database_Info
   R  Database name                 = *OTHERDBS
   R  Outbound protocol from router = DRDA
   R  Inbound protocol to router    = DB2RA
   R  Authenticate at gateway       = 0
   O  Parameter string              =
   R  DB_Database_Locator_Name      = /.../CELL_TORONTO/subsys/database/OTH_INST
```

The database name *OTHERDBS is a special value that identifies a common router used to access any target database not explicitly defined in the routing information object.

Here is an example of the DCE commands to create the object. The backslash (\) character is a continuation character.

Before any objects can be created, the DCE administrator needs to do the steps described in “DCE Administrator Tasks” on page 666.

First you must type the following in a file called *cdscp.inp*:

```
create object /.:/subsys/database/ROUTE1

add object /.:/subsys/database/ROUTE1 DB_Object_Type = R
add object /.:/subsys/database/ROUTE1 DB_Comment     = Routing_for_client_group_1
add object /.:/subsys/database/ROUTE1 DB_Target_Database_Info = \
/.../CELL_TORONTO/subsys/database/MVSDB;\
drda;db2ra;1;NOMAP,D,INTERRUPT_ENABLE;\
/.../CELL_TORONTO/subsys/database/GW_INST
add object /.:/subsys/database/ROUTE1 DB_Target_Database_Info = \
*OTHERDBS;drda;db2ra;0;;\
/.../CELL_TORONTO/subsys/database/OTH_INST
```

Then you must run either

- dcelogin principal password (on OS/2); or,
- dce_login principal password (on UNIX).

This should be followed by

- cdscp < cdscp.inp

Use the following command to display the object:

```
cdscp show object /.:/subsys/database/ROUTE1
```

For more information about the DCE commands, refer to the following DCE publications:

- *DCE Administration Guide*
- *DCE Administration Reference*

Attributes of Each Object Class

In the DCE environment, each object and object attribute is identified by an object ID (OID). Each OID is obtained from a hierarchy of allocation authorities, where the highest authority is the International Organization for Standardization (ISO).

Table 40 on page 652 shows the attributes for each object class and Table 41 on page 652 shows their attributes.

Table 40. Object Attribute Classes

Object Class	Object ID (OID)	Required Attributes	Optional Attributes
(DB) Database_Object	1.3.18.0.2.6.12	DAU, DOT, DDP, DNN	DCO, DPN, DRL, DLN, DCP, DPR
(DL) Database_Locator_Object	1.3.18.0.2.6.13	DOT, DCP	DCO, DPN, DRL
(RI) Routing_Information_Object	1.3.18.0.2.6.14	DOT, DTI	DCO, DPN, DRL

Table 41. Object Class Attributes

Attribute Name	OID	Minimum Length	Maximum Length	Syntax
(DAU) DB_Authentication	1.3.18.0.2.4.39	1	1024	Char
(DCO) DB_Comment	1.3.18.0.2.4.30	1	1024	Char
(DCP) DB_Communication_Protocol	1.3.18.0.2.4.31	1	1024	Char
(DDP) DB_Database_Protocol	1.3.18.0.2.4.32	1	1024	Char
(DLN) DB_Database_Locator_Name	1.3.18.0.2.4.33	1	1024	Char
(DNN) DB_Native_Database_Name	1.3.18.0.2.4.34	1	1024	Char
(DOT) DB_Object_Type	1.3.18.0.2.4.35	1	1	Char
(DPN) DB_Product_Name	1.3.18.0.2.4.36	1	1024	Char
(DRL) DB_Product_Release	1.3.18.0.2.4.37	1	1024	Char
(DTI) DB_Target_Database_Info	1.3.18.0.2.4.38	1	1024	Char
(DPR) DB_Principal	1.3.18.0.2.4.63	1	1024	Char
Note: Multiple values are allowed for DCP, DDP, and DTI. Only one value is allowed for the other attributes.				

Details About Each Attribute

The following section describes each attribute.

Note: DCE Directory Services does not check that the entries are valid for DB2. Ensure that you enter the attributes that are required and that you enter the correct values.

DB_Authentication (DAU)

Authentication method required by the object. This attribute is required for the database object of a DB2 server. The value must be CLIENT, SERVER, or DCE.

DB_Principal (DPR)

If authentication method is “DCE,” enter the DCE principal in this attribute.

DB_Comment (DCO)

For documentation purposes only.

DB_Communication_Protocol (DCP)

A multi-value attribute where each value consists of tokens that describe the network protocol supported. Examples of the network protocols are TCP/IP, APPC, IPX/SPX, and NetBIOS. (These last two are appropriate for OS/2 only.) Each token is separated by a semicolon. Do not put spaces between the tokens.

- The tokens for TCP/IP are:

  1. tcpip
  2. Host name of the target node
  3. Port number used by the object to listen for incoming TCP/IP connect requests
  4. (Optional) Security can be either NONE or SOCKS.

  For example: `tcpip;HOSTNAME;1234`

- The tokens for APPC are:

  1. appc
  2. Network ID of the target to which to object belongs.
  3. LU name where the target can be found.
  4. Transaction Program Name (TPN) representing the object in the LU (For DB2 for MVS/ESA, use DB2DRDA as the TPN.)
  5. Mode name
  6. Type of security used by the target. The values are:
     – NONE
     – PROGRAM
     – SAME

  For example: `appc;NETID;TARGETLU;TPNAME;MODE;PROGRAM`

  Note: For APPC, the client must use its local control point (CP) as its LU name.

- (OS/2 only) The tokens for IPX/SPX are:

  1. ipxspx
  2. Name of the file server
  3. Name of the object

  For example: `ipxspx;SVR_NAME;OBJ_NAME`

- (OS/2 only) The tokens for NetBIOS are:

  1. netbios
  2. Node name of the server

  For example: `netbios;SVR_NNME` where the client adapter name is found in either the environment variable *db2clientadpt* or the database manager configuration parameter *dft_client_adpt*.

DB_Database_Protocol (DDP)

The database protocol or protocols supported by the target database. Examples of the values are DB2RA and DRDA. The following are the *cdscp* commands to add two protocols.

```
add object /.:/subsys/database/AIXDB1 DB_Database_Protocol db2ra
add object /.:/subsys/database/AIXDB1 DB_Database_Protocol drda
```

DB_Database_Locator_Name (DLN)

The DCE name of the database locator object. In the database object, the name is for the DBMS instance. In the routing information object, the name is for the DB2 Connect instance.

For example, `/.../CELL_TORONTO/subsys/database/AIX_INST`

DB_Native_Database_Name (DNN)

The database name or alias by which the database is known within the instance containing the database. This is the name that a local application on the instance uses to connect to that database.

The name is up to 8 characters for a DB2 for Universal Database database. For other databases, the length of the name may be different. For example it can be up to 18 characters for databases on DB2 for MVS/ESA.

DB_Object_Type (DOT)

The type of object. This attribute is required for all objects and can be one of the following:

D Database object
L Database locator object
R Routing information object

DB_Product_Name (DPN)

The identification of the product. For documentation purposes only.

DB_Product_Release (DRL)

The product release level. For documentation purposes only.

DB_Target_Database_Info (DTI)

A multi-value attribute where each value consists of a fixed number of tokens, separated by a semicolon. Do not put spaces between the tokens. The tokens must be in the following order:

1. Database name. The DCE name of a target database for which the routing service is provided. The value *OTHERDBS specifies a default gateway for any target databases not explicitly defined in the routing information object.
2. Outbound protocol from router. The database protocol used by the target database, or the database protocol the routing DB2 Connect instance uses to communicate with that target database. For example, DRDA.
3. Inbound protocol to router. The database protocol accepted by the routing DB2 Connect instance object. For example, DB2RA.

4. Authenticate at gateway. The valid values are 0 or 1. See Table 42 on page 656 for more details.
5. Parameter string which contains information specific to the DB2 Connect gateway. The string contains tokens that must be in the order described below. The tokens are separated by commas. For tokens that are not specified, the default is used.
   - Map-file name. The fully-qualified name of the SQLCODE mapping file that overrides the default SQLCODE mapping. To turn off SQLCODE mapping, specify NOMAP.
   - D. The application disconnects from the DRDA server database when specific SQLCODEs are returned. Refer to the *DB2 Connect User's Guide* for details about the SQLCODEs.
   - INTERRUPT_ENABLED. DB2 Connect will drop the connection and roll back the unit of work when a client issues an interrupt while connected to the DRDA server.

   The following are some examples:

   ```
   NOMAP
   /u/username/sqllib/map/dcs1new.map,D
   /u/username/sqllib/map/dcs1new.map,D,INTERRUPT_ENABLED
   ```

   Where defaults are used, use a comma to preserve the order of the tokens, for example:

   ```
   ,D
   ```

   or

   ```
   ,,INTERRUPT_ENABLED
   ```

   Refer to the *DB2 Connect User's Guide* for details about the Parameter string.
6. The DCE name of the DB2 Connect instance that provides the routing service.

The following is an example of the DB_Target_Database_Info:

```
/.../CELL_TORONTO/subsys/database/MVSDB;\
drda;db2ra;0;;\
/.../CELL_TORONTO/subsys/database/GW_INST
```

Note: In the above example, the back slash (\) is a line continuation character.

Directory Services Security

When using DCE directory services in an environment without a DB2 Connect gateway, authentication is the same as is used for other DB2 Client Application Enabler accessing database servers. For more information, see “Authentication” on page 111.

When using DCE directory services in an environment with a DB2 Connect gateway, the DB2 Connect administrator determines where user names and passwords are validated. With DCE directories, specify the following:

- The security type of the communication protocol in the database locator object representing the DB2 Connect workstation. (If a remote client is connected to the DB2 Connect Extended Edition gateway via an APPC connection, specify a security type of NONE in the *DCE Locator Object* of the gateway.)
- The authentication type in the database object.
- The security type of the communication protocol in the database object (or its associated locator object).
- The authenticate at gateway token in the routing information object.

Table 42 shows the possible combinations of these values and where validation is performed for each combination using APPC connections. The combinations shown in this table are supported by DB2 Connect with DCE Directory Services.

Table 42. Valid Security Scenarios with DCE using APPC Connections

	Database Object of the Server		**Routing Object**	**Validation**
Case	**Authentication**	**Security**	**Authenticate at Gateway**	
1	CLIENT	SAME	0	Remote client (or DB2 Connect workstation)
2	CLIENT	SAME	1	DB2 Connect workstation
3	SERVER	PROGRAM	0	DRDA server
4	SERVER	PROGRAM	1	DB2 Connect workstation and DRDA server
5	DCE	NONE	NOT APPLICABLE	DCE

Table 43 shows the possible combinations of these values and where validation is performed for each combination using TCP/IP connections. The combinations shown in this table are supported by DB2 Connect with DCE Directory Services.

Table 43. Valid Security Scenarios with DCE using TCP/IP Connections

Case	**Authentication**	**Authenticate at Gateway**	**Validation**
1	CLIENT	0	Client
2	CLIENT	1	DB2 Connect workstation
3	SERVER	0	DRDA server
4	NOT APPLICABLE	NOT APPLICABLE	None
5	DCE	NOT APPLICABLE	DCE

Each combination is applicable to both APPC and TCP/IP and is described in more detail below:

1. The user name and password are validated only at the remote client. (For a local client, the user name and password are validated only at the DB2 Connect workstation.)

The user is expected to be authenticated at the location he or she first signs on to. The user ID is sent across the network, but not the password. Use this type of security only if all client workstations have adequate security facilities.

2. The user name and password are validated at the DB2 Connect workstation only. The password is sent across the network from the remote client to the DB2 Connect workstation but not to the DRDA server.
3. The user name and password are validated at the DRDA server only. The password is sent across the network from the remote client to the DB2 Connect workstation and from the DB2 Connect workstation to the DRDA server.
4. The user name and password are validated at both the DB2 Connect workstation and the DRDA server. The password is sent across the network from the remote client to the DB2 Connect workstation and from the DB2 Connect workstation to the DRDA server.

   Because validation is performed in two places, the same set of user names and passwords must be maintained at both the DB2 Connect workstation and the DRDA server.
5. A DCE token is obtained from the DCE Security Server.

Notes:

1. For AIX-based systems, all users using security type `SAME` must belong to the AIX system group.
2. For AIX-based systems with remote clients, the instance of the DB2 Connect product running on the DB2 Connect workstation must belong to the AIX system group.
3. Access to a DRDA server is controlled by its own security mechanisms or subsystems; for example, the Virtual Telecommunications Access Method (VTAM) and Resource Access Control Facility (RACF). Access to protected database objects is controlled by the SQL **GRANT** and **REVOKE** statements.

Configuration Parameters and Environment Variables

The following configuration parameters are used with DCE directories. An example of the values is shown. Refer to “Distributed Services” on page 549 for details.

- *dir_obj_name* is the database instance name which is concatenated with *dir_path_name*. If the instance name is used as the target of the ATTACH command, the name must be less than or equal to 8 characters and all upper case, for example:

  ```
  AIX_INST
  ```

- *dir_type* identifies whether or not to use DCE directory services. To enable DCE directory services, this parameter must be set to:

  ```
  DCE
  ```

  Note that *dir_type* is set to NONE and cannot be updated on database clients that do not support the use of DCE directory services.

- *dir_path_name* is the directory path name provided by the DCE administrator, for example:

  ```
  /.:/subsys/database/
  ```

- *route_obj_name* is an optional parameter that provides the name of the routing information object. The name can be fully-qualified, for example:

  ```
  /.:/subsys/database/ROUTE1
  ```

  or a one-part name that will be concatenated with *dir_path_name*, for example:

  ```
  ROUTE1
  ```

- *dft_client_comm* is an optional parameter that specifies the communications protocol used by the client, for example:

  ```
  TCPIP
  ```

  This parameter can also specify more than one protocol, for example:

  ```
  TCPIP,APPC   (on UNIX-based platforms)
  TCPIP,APPC,IPXSPX,NETBIOS    (on OS/2 platforms)
  ```

- *dft_client_adpt* is an optional parameter that specifies the default client adapter number for the NetBIOS protocol on OS/2. The valid range of numbers is zero through fifteen (0 to 15). If this parameter contains a non-numeric value, then the value defaults to zero (0). If this parameter contains a value outside the range allowed, then the value defauts to zero (0).

For the following parameters, environment variables can override the values.

Configuration Parameter	**Environment Variable**
dir_path_name	DB2DIRPATHNAME
route_obj_name	DB2ROUTE
dft_client_comm	DB2CLIENTCOMM
dft_client_adpt	DB2CLIENTADPT

The rules for setting these environment variables is the same as their corresponding configuration parameter. For example, like the *dft_client_comm* parameter, the DB2CLIENTCOMM is a character string that can have multiple values, each separated by a comma, for example:

```
export DB2CLIENTCOMM=TCPIP,APPC
```

CATALOG, CONNECT, and ATTACH Commands

DCE information needs to be specified in the following commands:

- CATALOG GLOBAL DATABASE Command
- CONNECT Command
- ATTACH Command

CATALOG GLOBAL DATABASE Command

Use the CATALOG GLOBAL DATABASE command when the client and server have a different path name, or when the database name contains more than 8 characters or mixed case characters. The database administrator enters the DCE name of the database and directory type DCE.

For example:

- When the path names are different, for example if *dir_path_name = /.../CELL_TORONTO/subsys/database/*:

```
CATALOG GLOBAL DATABASE
/.../CELL_VANCOUVER/subsys/database/VMDB AS VANVMDB
USING DIRECTORY DCE WITH "comment-string"
```

- When the database name contains more than 8 characters, such as the name DB_LONGNAME:

```
CATALOG GLOBAL DATABASE
/.../CELL_VANCOUVER/subsys/database/DB_LONGNAME AS VANVMDB
USING DIRECTORY DCE WITH "comment-string"
```

CONNECT Command

To retrieve the appropriate DCE directory object, the client must know the fully-qualified DCE name of the database or the DBMS instance. Some of the methods of specifying the name in the CONNECT command follow.

- Enter the alias, for example:

```
CONNECT TO VANVMDB
```

- Enter the one-part name, for example:

```
CONNECT TO VMDB
```

In this case, the path name specified at the client must be the same as the path name specified at the server. (The path name is specified by the *dir_path_name* configuration parameter or the corresponding environment variable.)

ATTACH Command

The effective path name of the client must be the same as the path name of the target DBMS instance.

If the *dir_path_name* is the same for client and server (for example, `/.../CELL_TORONTO/subsys/database/`) and the *dir_obj_name* at the database server is `AIX_INST`, the command to attach to the instance is:

```
ATTACH TO AIX_INST
```

How a Client Connects to a Database

Figure 63 shows a sample configuration of a database network with two DCE cells. /.../CELL_TORONTO and /.../CELL_VANCOUVER are the names of the cells. (Each of these cells contains a directory called /.:/subsys/database/ and while not illustrated in diagram, is used in other examples.)

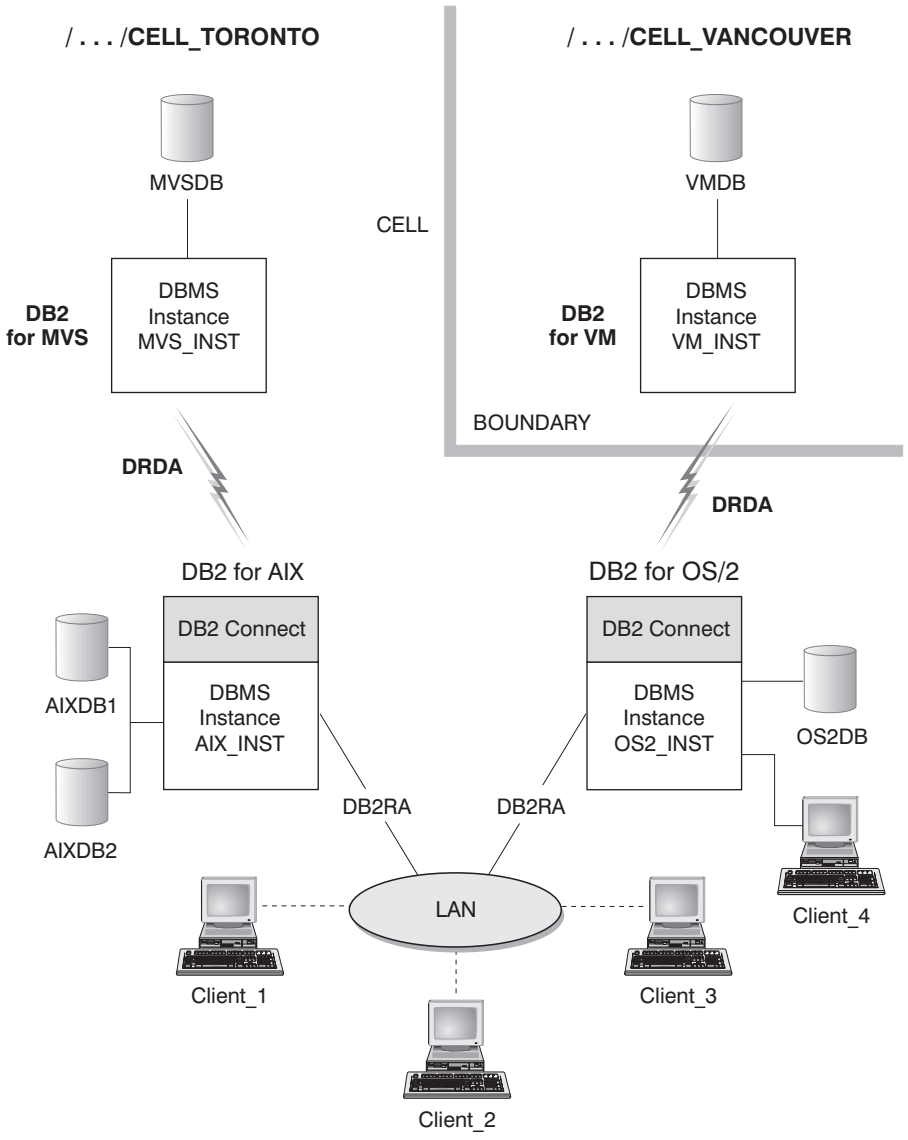


Figure 63. Configuration of A Network Database

To allow the clients in the TORONTO cell to access all the databases in both cells, values must be specified in the database manager configuration parameters and the following objects must be created:

- A database object for each database.
- A database locator object for the two database servers for DB2 for AIX and DB2 for OS/2.
- A single routing information object that is known to all clients. The attributes specify which DB2 Connect node to use for the MVSDB and VMDB databases.

The following provide examples of how a client connects to a database:

- Connecting to Databases in the Same Cell
- Connecting to a Database in a Different Cell.

These examples include the database manager configuration parameters that must be specified.

Connecting to Databases in the Same Cell

This section describes several examples of how clients connect to databases in the same cell.

1. Client_1 connects to AIXDB2. The database shares the same directory path name as the client.

   The database administrator needs to:
   - Specify the directory path name value in the configuration parameter *dir_path_name* (or the DB2DIRPATHNAME environment variable).
   - Specify the directory services type value to be DCE in the configuration parameter *dir_type*.
   - Specify the communication protocol in the configuration parameter *dft_client_comm* (or the DB2CLIENTCOMM environment variable).

   The local system database directory does not contain AIXDB2, so the DCE directory is searched using the fully-qualified name. The name is created by concatenating the value for the configuration parameter *dir_path_name* (or the DB2DIRPATHNAME environment variable) with AIXDB2.

   The sequence of events is:
   a. Client_1 obtains the database object for AIXDB2 using the DCE name of the database `/.../CELL_TORONTO/subsys/database/AIXDB2`.
   b. From this object, Client_1 knows that AIXDB2 uses the DB protocol DB2RA, which is the same protocol that Client_1 uses.
   c. The DB protocols match, so Client_1 reads the DBMS locator object for AIX_INST, retrieves the communications protocol attribute value that matches the one it uses, and uses the information to start a conversation with that DBMS instance.
2. Client_3 connects to MVSDB. The database shares the same directory path name as the client and uses a different database protocol from the client.

   The database administrator needs to:

- Specify the directory path name value in the configuration parameter *dir_path_name* (or the DB2DIRPATHNAME environment variable).
- Specify the directory services type value to be DCE in the configuration parameter *dir_type*.
- Specify the communication protocol in the configuration parameter *dft_client_comm* (or the DB2CLIENTCOMM environment variable).
- Specify the DCE name of the default routing information object in the configuration parameter *route_obj_name* (or the DB2ROUTE environment variable).

The sequence of events is:

a. Client_3 obtains the database object for MVSDB using the DCE name of the database `/.../CELL_TORONTO/subsys/database/MVSDB`.
b. From this object, Client_3 finds that MVSDB only uses the DB protocol DRDA, which is not the protocol that Client_3 uses.
c. Client_3 then obtains the routing information object using the name defined in the *route_obj_name* configuration parameter or the DB2ROUTE environment variable. The client finds the target database information for MVSDB.
d. Client_3 reads the database locator object associated with the MVSDB target database information, retrieves the communication protocol, and sends an SQL CONNECT request to the router.
e. The router then sets up an APPC connection with MVSDB.

Connecting to a Database in a Different Cell

This section describes an example of how a client connects to a database in a different cell when the database protocols are different.

1. Client_3 has previously been configured to use the following:
   - DCE directory services, by specifying DCE for the *dir_type* parameter.
   - A cell other than CELL_VANCOUVER through the configuration parameter *dir_path_name*, for example:

     ```
     /.../CELL_TORONTO/subsys/database/
     ```

2. In order for Client_3 to connect to VMDB, the database administrator needs to:
   - Explicitly catalog VMDB in the local system database directory. Associate the DCE name for VMDB with a locally unique database alias, and issue the CONNECT statement with the alias value. For example:

     ```
     CATALOG GLOBAL DATABASE
     /.../CELL_VANCOUVER/subsys/database/VMDB AS VANVMDB
     USING DIRECTORY DCE WITH "comment-string"
     ```

     followed by:

     ```
     CONNECT TO VANVMDB
     ```

   - Specify the communication protocol in the configuration parameter *dft_client_comm* (or the DB2CLIENTCOMM environment variable).
   - Specify the DCE name of the default routing information object in the configuration parameter *route_obj_name* (or the DB2ROUTE environment variable).

The sequence of events is:

a. Client_3 finds the fully qualified DCE name of VANVMDB in its system database directory.
b. Client_3 obtains the database object for VMDB using the DCE name of the database `/.../CELL_VANCOUVER/subsys/database/VMDB`.
c. From this object, Client_3 finds that VMDB only uses the DB protocol DRDA, which is not the protocol that Client_3 uses.
d. Client_3 then obtains the routing information object using the name defined in the *route_obj_name* configuration parameter or the DB2ROUTE environment variable. The client finds the target database information for VMDB.
e. Client_3 reads the database locator object associated with the VMDB target database information and retrieves the communication protocol and sends an SQL CONNECT request to the router.
f. The router then sets up an APPC connection with VMDB.

How Directories are Searched

If the DCE directory is used in an environment where all the target databases share the same directory path name, no local directories are required on the clients.

This section describes the order in which directories are searched for the following:

- ATTACH Command
- CONNECT Command

ATTACH Command

Figure 64 on page 664 shows how the directories are searched when a client attaches to a DBMS instance called ABC_INST.

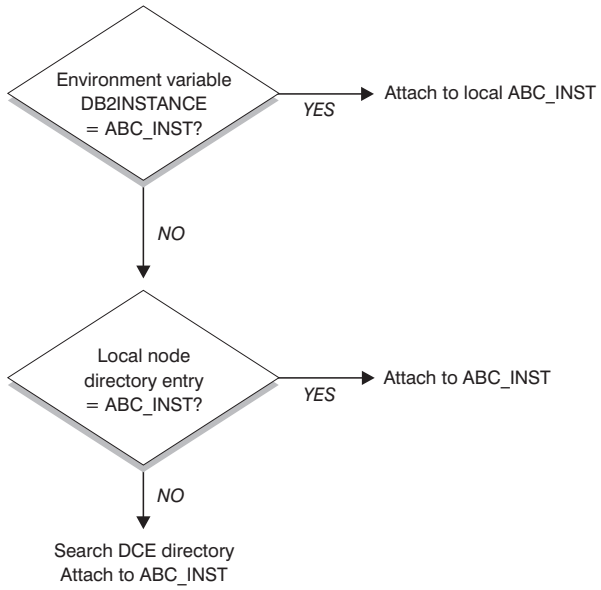


Figure 64. How Directories are Used to Attach a Database

CONNECT Command

Figure 65 on page 665 shows how the directories are searched when a client connects to a database called DBTEST.

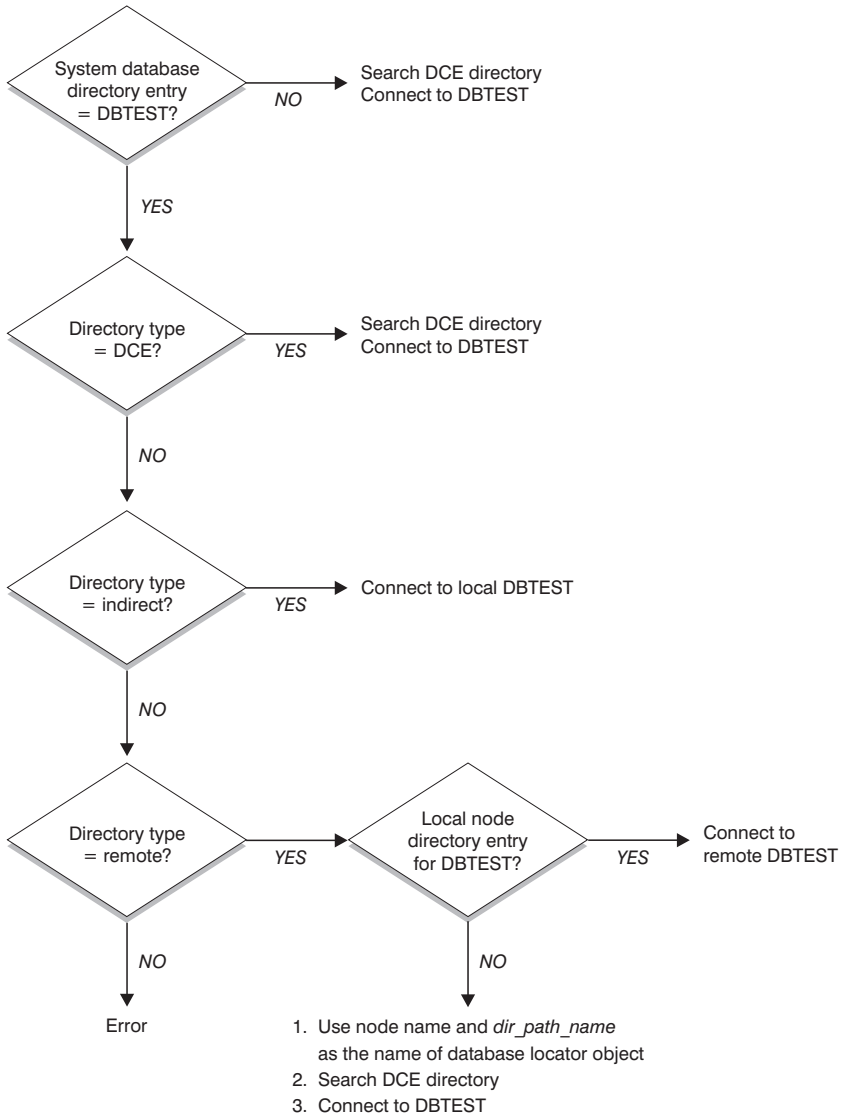


Figure 65. How Directories are Used to Connect a Database

Temporarily Overriding DCE Directory Information

You can use the local database directory to override the DCE directory information. For example, if you CONNECT TO DBTEST where `/.:/subsys/database/DBTEST` is defined in the DCE directory as residing on a host called JAGUAR, you can temporarily change

DBTEST to a different database residing on a host called STORM. Catalog DBTEST locally as a remote database with a node directory entry pointing to STORM.

You can create an alias for a database whose DCE name does not follow the directory path name of the client. See “CATALOG GLOBAL DATABASE Command” on page 659 for details about the command.

Directory Services Tasks

The tasks that must be performed to setup and use DCE Directory Services are listed below. The following sections describe the details of each task.

- DCE Administrator Tasks

  The DCE administrator must update the DCE directory so that the new database resource information can be added.
- Database Administrator Tasks

  The database administrator must update the DCE directory and supply information for DB2 installation and configuration.
- Database User Tasks

  The database user must log in to DCE and know the target database name.

In addition, the network administrator sets up the network access for each user node. Refer to the network documentation for the details.

DCE Administrator Tasks

The DCE administrator must do the following tasks before the directory objects can be created or read:

- Assign the directory subtree for DB2, for example /.:/subsys/database
- Grant the privileges to the database administrator to create directory objects
- Grant the privileges to the database users to read the directory objects
- Add the information for the new DCE directory object attributes to the *DCE attribute table*.

  Edit the CDS attributes file (on UNIX platforms */etc/dce/cds_attributes*; on OS/2 *X:**opt**dcelocal**etc**cds_attr*, where "X" is the appropriate drive) and append the following:

```
1.3.18.0.2.4.30 DB_Comment                char
1.3.18.0.2.4.31 DB_Communication_Protocol char
1.3.18.0.2.4.32 DB_Database_Protocol      char
1.3.18.0.2.4.33 DB_Database_Locator_Name  char
1.3.18.0.2.4.34 DB_Native_Database_Name   char
1.3.18.0.2.4.35 DB_Object_Type            char
1.3.18.0.2.4.36 DB_Product_Name           char
1.3.18.0.2.4.37 DB_Product_Release        char
1.3.18.0.2.4.38 DB_Target_Database_Info   char
1.3.18.0.2.4.39 DB_Authentication         char
1.3.18.0.2.4.63 DB_Principal              char
```

- Ensure DCE is running when users need access to the databases using DCE Directory Services.

For more information, refer to the DCE documentation for the platform you are using.

Database Administrator Tasks

The database administrator must do the following tasks:

- Obtain the directory subtree for the database resources from the DCE administrator. For example, `/.:/subsys/database`
- During installation of the DB2 database manager, ask the DCE administrator to add the new DCE directory object attributes required by DB2.
- Assign a unique name for each DBMS instance in the DCE directory subtree. For example, `/.:/subsys/database/AIX_INST`
- For each DBMS instance specify the database manager configuration parameters for DCE.
  - *dir_type*
  - *dir_obj_name*
  - *dir_path_name*
  - *route_obj_name*
  - *dft_client_comm*
  - *dft_client_adpt*

  Some of the configuration parameters can be temporarily overridden by environment variables set by the client. Refer to “Configuration Parameters and Environment Variables” on page 657 for more information.
- Assign a unique name for each database in the DCE directory subtree. Specify the name in the *dir_obj_name* parameter in the database configuration file.
- Create the objects for DCE Directory Services using the DCE *cdscp* commands to create and display objects. The objects are created separately from the database manager installation process and the database manager instance start process.

  Three types of objects exist.
  - A database object is required for each target database.
  - A database locator object is required for each DB2 Connect instance and each DBMS instance (without DB2 Connect) which is associated with more than one database.
  - Routing information objects are required to access a host database.
- Depending on each environment, the database administrator must determine:
  - How to group the clients into logical groups considering what databases they access, and what communications protocols they use.
  - How many routing information objects are required.
  - Which target databases should be recorded in each routing information object.
  - Which routing information objects should be known to which group of clients.

  Refer to “Creating Directory Objects” on page 647 for details about the objects.

Database User Tasks

The database user must do the following tasks:

- Obtain the name of the database from the database administrator. This name can be a simple one-part name, or a fully-qualified DCE name.
- If needed, specify the values required for DCE Directory Services in the environment variables. Environment variables set by the client can temporarily override the configuration parameters.
  - If host database access is required, obtain the fully-qualified DCE name of the routing information object from the database administrator. If this name is not specified in the *route_obj_name*, or it is a different name, specify this name in the DB2ROUTE environment variable before trying to connect to the host database.
  - If your preferred communication protocol is not specified in *dft_client_comm*, or it is a different protocol, specify the communication protocol for the client in the DB2CLIENTCOMM environment variable. Here are **some** UNIX examples:

    ```
    export DB2CLIENTCOMM=tcpip
    export DB2CLIENTCOMM=appc
    export DB2CLIENTCOMM=tcpip,appc
    export DB2CLIENTCOMM=appc,tcpip
    ```

    Some OS/2 examples are:

    ```
    export DB2CLIENTCOMM=ipxspx
    export DB2CLIENTCOMM=netbios
    export DB2CLIENTCOMM=tcpip,ipxspx,netbios
    export DB2CLIENTCOMM=netbios,tcpip,ipxspx,appc
    ```

    If more than one communication protocol exists, the first one specified is used.
- If any of the databases has a DCE name that is not in the directory path defined in the *dir_path_name* configuration parameter or the DB2DIRPATHNAME environment variable, then explicitly catalog the database with the CATALOG GLOBAL DATABASE command. Refer to "CATALOG GLOBAL DATABASE Command" on page 659 for more information.
- Log in to DCE before connecting to the target database or attaching to the database instance. Refer to the *OSF DCE Administration Guide* for more information about the login command.

Directory Services Restrictions

This section describes what is not supported.

- Not all database clients may be supported. See your *Quick Beginnings* manual to determine whether DCE directory services is supported from your DB2 client. Currently, support is only provided for DB2 Client Application Enabler for all UNIX platforms (except HP-UX) and for OS/2.
- A client cannot use DCE Directory Services to connect to a DB2 for OS/2 Version 1 server.
- Only OS/2 clients can use any or all of the TCP/IP, APPC, NetBIOS, and IPX/SPX protocols. All supported UNIX clients (see above) can only use the TCP/IP and APPC protocols.

- LIST DATABASE (or NODE) DIRECTORY COMMANDS only provide entries from the local directories and not entries from the DCE directory. You can use the *cdscp show object* command in DCE to display the objects.
- When all of the following conditions exist, the owner of the database manager instance must login to DCE before starting the database manager (using the `db2start` command).
  - The database manager instance is configured to support DCE directory services through the *dir_type* configuration parameter
  - The cell directory services object can only be read by explicitly logging into DCE
  - The DCE directory must be accessed to support either of the following:
    - A transaction manager database (specified by the *tm_database* configuration parameter) located on another instance
    - A client that cannot support DCE directory services, or is not configured to use DCE directory services.

  Note: When performing the DCE login, you should use a principal that has a long ticket lifetime.
- When using a DDCS Version 2.2 (or earlier) gateway to connect a client that is using DCE directory services to a DRDA server, you must catalog the database alias in the gateway's local directory. This database alias must be the same as the alias on the client and it must represent the same database.

Appendix G. X/Open Distributed Transaction Processing Model

The following figure illustrates the X/Open Distributed Transaction Processing (DTP) model and the relationship between the three components included in this model.

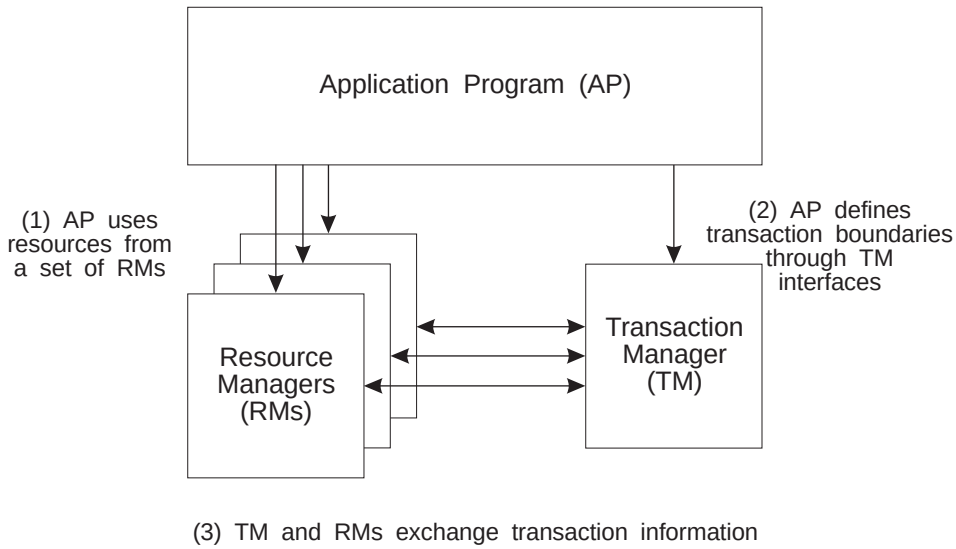


Figure 66. X/Open Distributed Transaction Processing (DTP) Model

The following sections provide an overview of each of the components included in the Distributed Transaction Processing model:

- Application Program (AP)
- Transaction Manager (TM)
- Resource Managers (RM).

"XA Function Supported" on page 674 provides additional information about the XA support provided by DB2.

Application Program (AP)

The application program (AP) defines transaction boundaries, and specifies the application-specific actions that make up the transaction.

For example, a CICS* application program might want to access resource managers (RMs) such as a database and a CICS Transient Data Queue, and use programming logic between these accesses to manipulate the data. Each access request is passed

to the appropriate resource managers through function calls specific to that RM. In the case of DB2, these could be function calls generated by the DB2 precompiler for each SQL statement, or database calls coded directly by the programmer using the APIs.

A transaction manager product usually includes a transaction processing (TP) Monitor to run the user's application. The TP Monitor provides APIs to allow an application to start and end a transaction, and to perform application scheduling and load balancing among the many users who want to run the application. Therefore the application program (AP) in a DTP environment is really a combination of both the user application and the TP monitor.

To facilitate an efficient online transaction processing (OLTP) environment, the TP Monitor pre-allocates a number of server processes at startup, and then schedules and reuses them among the many user transactions. This saves on the amount of system resources by allowing more concurrent users to be supported with a smaller number of server processes and their corresponding RM processes. Reusing these processes also avoids the overhead of starting up a process in the TM and RMs for each user transaction or program. (A program invokes one or more transactions.) This also means the server processes are the actual "user processes" to the TM and the RMs. This has implications for security administration and application programming. See “Security Considerations” on page 259 for details.

The following types of transactions are possible from a TP Monitor:

- Non-XA transactions

  These transactions involve RMs that are not defined to the TM, and are therefore not coordinated under the two-phase commit protocol of the TM. This might be necessary if the application needs to access an RM that does not support the XA interface. The user basically just uses the TP monitor as a mechanism that provides efficient scheduling of applications and load balancing. Since the TM does not explicitly "open" the RM for XA processing, the RM treats this application as any other application that runs in a non-DTP environment.
- Global transactions

  These transactions involve RMs that are defined to the TM, and are under the TM's two-phase commit control. A global transaction is a unit of work that could involve one or more RMs. A ***transaction branch*** is the part of work between a TM and an RM to support the global transaction. A global transaction could have multiple transaction branches when multiple RMs are accessed through one or more application processes that are coordinated by the TM.

  Loosely coupled, global transactions exist when each of a number of application processes accesses the RMs as if they are in a separate global transaction, but those applications are under the coordination of the TM. Each application process will have its own transaction branch within an RM. When a commit or rollback is requested by any one of the APs, TM, or RMs, the transaction branches are completed altogether. It is the application's responsibility to ensure that resource deadlock does not occur among the branches. (Note that the transaction coordination performed by the DB2 transaction manager for applications prepared

with the SYNCPOINT(TWOPHASE) option is roughly equivalent to these global, loosely-coupled transactions. See “Updating Multiple Databases” on page 242.)

Tightly coupled global transactions exist when multiple application processes take turns to do work under the same transaction branch in an RM. To the RM, the two application processes are treated as a single entity. The RM must ensure that resource deadlock does not occur within the transaction branch.

Transaction Manager (TM)

The transaction manager (TM) assigns identifiers to transactions, monitors their progress, and takes responsibility for transaction completion and failure. The transaction branch identifiers (known as XIDs) are assigned by the TM to identify both the global transaction and the specific branch within an RM. This is the correlation token between the log in a TM and the log in an RM. The XID is needed for two-phase commit, or rollback, to perform the **resynchronization** operation (also known as **resync**) on a system startup, or to let the administrator perform a **heuristic** operation (also known as **manual intervention**) if necessary.

After a TP Monitor is started up, it will ask the TM to open all the RMs that a set of application servers have defined. The TM will pass the ***xa_open*** calls to the RMs so that they can be initialized for DTP processing. As part of this startup procedure, the TM will perform the resync to recover all **indoubt transactions**. An indoubt transaction is a global transaction that was left in an uncertain state. This occurs when either the TM or at least one RM becomes unavailable after successfully completing the first phase (that is, the prepare phase) of the two-phase commit protocol. The RM will not know whether to commit or rollback its branch of the transaction until the TM can consolidate its own log with the RMs' when they become available again. To perform the resync operation, the TM will issue the ***xa_recover*** call one or more times to each of the RMs to identify all the indoubt transactions. The TM will compare the replies with the information in its own log to determine whether it should inform the RMs to ***xa_commit*** or ***xa_rollback*** those transactions. If an RM had already committed or rolled back its branch of an indoubt transaction through a heuristic operation by its administrator, the TM will issue an ***xa_forget*** call to that RM to complete the resync operation.

When a user application requests a commit or rollback, it must use the API provided by the TP Monitor or TM so that the TM can coordinate the commit and rollback among all the RMs involved. For example, when a CICS application issues the CICS SYNCPOINT request to commit a transaction, the CICS/6000* TM will in turn issue the XA calls such as ***xa_end, xa_prepare, xa_commit, or xa_rollback*** to request the RM to commit or rollback the transaction. The TM could choose to use one-phase instead of two-phase commit if only one RM is involved, or if an RM replies that its branch is read-only.

Resource Managers (RM)

A resource manager (RM) provides access to shared resources such as databases.

DB2 as a resource manager of a database resource can participate in a ***global transaction*** that is being coordinated by an XA-compliant TM. As required by the XA interface, the database manager provides a ***db2xa_switch*** external C variable of type ***xa_switch_t*** to return the XA switch structure to the TM. This data structure contains the addresses of the various XA routines to be invoked by the TM, and the operating characteristics of the RM. For more information on the XA functions supported by the database manager see "XA Function Supported."

There are two methods for the RM to register its participation in each global transaction: **static registration** and **dynamic registration**. The database manager implements the more advanced and efficient dynamic registration method:

- Static registration requires the TM to issue for every transaction the ***xa_start, xa_end, xa_prepare*** series of calls to all the RMs defined for the server application regardless whether this particular RM is used by the transaction or not. This is inefficient when not every transaction involves every RM. This inefficiency gets worse if there are many RMs defined.
- Dynamic registration is provided by the XA specification for flexibility and efficiency. An RM will register to the TM using the ***ax_reg*** call only when the RM receives a request for its resource. Note that there is no performance disadvantage with this method even when there is only one RM defined, or when every RM is used by every transaction because the ***ax_reg*** and ***xa_start*** calls have similar paths in the TM.

The XA interface provides two-way communication between a TM and an RM. It is a system-level interface between the two DTP software components, not an ordinary application program interface to which an application developer codes. However, application developers should be familiar with the programming function and restrictions that the DTP software components impose. See the *Embedded SQL Programming Guide* for information about the X/Open XA interface programming considerations.

Although the XA interface is invariant, each XA-compliant TM may have product specific ways of integrating an RM. For information about integrating your DB2 product as a resource manager with a specific transaction manager, see the appropriate TM product documentation.

XA Function Supported

The common server versions of DB2 support the XA91 specification defined in *X/Open CAE Specification Distributed Transaction Processing: The XA Specification* manual, with the following exceptions:

- Asynchronous services.

  The XA specification allows the interface to use asynchronous services where the result of a request can be checked at some later time. The database manager requires the use of the requests to be invoked in synchronous mode.
- Static registration.

  The XA interface allows for two ways to register an RM: static registration and dynamic registration. The database manager implements dynamic registration

which is more advanced and efficient. Refer to “Resource Managers (RM)” on page 673 for more details about these two methods.

XA Switch Usage

As required by the XA interface, the database manager provides a ***db2xa_switch*** external C variable of type ***xa_switch_t*** to return the XA switch structure to the TM. Other than the addresses of the various XA functions, the following fields are returned:

Field	Value
name	The product name of the database manager: for example, DB2 for AIX
flags	TMREGISTER || TMNOMIGRATE Explicitly states that the TM should not use association migration or asynchronous operation. Also implicitly states that dynamic registration is used by this RM.
version	Must be zero.

XA Open and Close Strings Usage

The database manager open string has the following syntax:

```
"database_alias<,username,password>"
```

The `database_alias` is required to specify the database alias name of the database. This alias name is the same as the database name unless you have explicitly cataloged an alias name after you created the database. The `username` and `password` are optional, and are used to provide authentication information to the database if the database is set up with **authentication=SERVER**.

The database manager does not use the XA close string and its content will be ignored.

DB2 for Windows NT supports the X/Open XA Interface. However the interface to the db2xa_switch data structure is different for DB2 for Windows NT due to operating system differences.

The pointer to the xa_switch structure, `db2xa_switch`, is exported as DLL data. This implies that a Windows NT application using this structure must reference it in one of two ways:

- Through one additional level of indirection. In a C program, this can be accomplished by defining the macro:

  ```
  #define db2xa_switch (*db2xa_switch)
  ```

  prior to an use of `db2xa_switch`,
- If using the Microsoft Visual C++ compiler, `db2xa_switch` can be defined as:

  ```
  extern __declspec(dllimport) struct xa_switch_t db2xa_switch
  ```

DB2 for Windows NT also provides an API, `db2xacic`, which returns the address of the `db2xa_switch` structure. This function is prototyped as:

```
struct xa_switch_t * SQL_API_FN  db2xacic( )
```

The following code illustrates the different ways the `db2xa_switch` can be accessed via a C program:

```
#include <stdio.h>
#include <xa.h>

struct xa_switch_t * SQL_API_FN  db2xacic( );

#ifdef DECLSPEC_DEFN
extern __declspec(dllimport) struct xa_switch_t db2xa_switch;
#else
#define db2xa_switch (*db2xa_switch)
extern struct xa_switch_t db2xa_switch;
#endif

main( )
{
   struct xa_switch_t *foo;
   printf ("%s \n", db2xa_switch.name );
   foo = db2xacic();
   printf ( "%s \n", foo->name );
   return ;
}
```

Making the Transacation Manager Known to the Resource Manager

DB2 needs to resolve the entry points to **ax_reg** and **ax_unreg** with the TM in order to be able to dynamically register a transaction:

- On UNIX platforms, this is done automatically when the application links in the DB2 and TM libraries.
- On OS/2, it is necessary for the RM to explicitly load the dynamic link library (DLL) that exports both these entry points at runtime, and to accomplish this the DLL name and path are retrieved from the *tp_mon_name* database manager configuration parameter. Ensure that this parameter is set appropriately when using the XA interface on an OS/2 platform.

Appendix H. Sample Tables

This appendix shows the information contained in the sample tables, and how to install and remove them. The sample tables are used in the examples that appear in this manual and other manuals in this library. In addition, the data contained in the sample files with BLOB and CLOB data types is shown.

The following sections are included in this appendix:.

In the sample tables, a question mark (-) indicates a null value.

The Sample Database

The examples in this book use a sample database. To use these examples, you must install the SAMPLE database. To use it, the database manager must be installed.

To Install the Sample Database

An executable file installs the sample database.[2] To install a database you must have SYSADM authority.

[2] For information related to this command, see the DB2SAMPL command in the *Command Reference*.

- **When Using UNIX-based Systems**

  If you are using the operating system command prompt, type:

  ```
  sqllib/misc/db2sampl <path>
  ```

  from the home directory of the database manager instance owner, where *path* is an optional parameter specifying the path where the sample database is to be created. Press Enter.[3] The schema of the authorization ID that invoked DB2SAMPL is the default schema.

- **When Using DB2 for OS/2**

  If you are using the operating system command prompt, type:

  ```
  db2sampl e
  ```

  where *e* is an optional parameter specifying the drive where the database is to be created. Press Enter.[4]

  If you are not logged on to your workstation through User Profile Management, you will be prompted to do so.

To Erase the Sample Database

If you do not need to access the sample database, you can erase it by using the DROP DATABASE command.

```
db2 drop database sample
```

CL_SCHED Table

Name:	CLASS_CODE	DAY	STARTING	ENDING
Type:	char(7)	smallint	time	time
Desc:	Class Code (room:teacher)	Day # of 4 day schedule	Class Start Time	Class End Time

DEPARTMENT Table

Name:	DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
Type:	char(3) not null	varchar(29) not null	char(6)	char(3) not null	char(16)
Desc:	Department number	Name describing general activities of department	Employee number (EMPNO) of department manager	Department (DEPTNO) to which this department reports	Name of the remote location

[3] If the path parameter is not specified, the sample tables are installed in the default path specified by the DFTDBPATH parameter in the database manager configuration file.

[4] If the drive parameter is not specified, the sample tables are installed on the same drive as DB2 for OS/2.

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

EMPLOYEE Table

Names:	EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
Type:	char(6) not null	varchar(12) not null	char(1) not null	varchar(15) not null	char(3)	char(4)	date
Desc:	Employee number	First name	Middle initial	Last name	Department (DEPTNO) in which the employee works	Phone number	Date of hire

JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
char(8)	smallint not null	char(1)	date	dec(9,2)	dec(9,2)	dec(9,2)
Job	Number of years of formal education	Sex (M male, F female)	Date of birth	Yearly salary	Yearly bonus	Yearly commission

See the following page for the values in the EMPLOYEE table.

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
char(6) not null	varchar(12) not null	char(1) not null	varchar(15) not null	char(3)	char(4)	date	char(8)	smallint not null	char(1)	date	dec(9,2)	dec(9,2)	dec(9,2)
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-24	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	500	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370	500	2030

EMPNO	FIRSTNME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIREDATE	JOB	ED LEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

EMP_ACT Table

Name:	EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
Type:	char(6) not null	char(6) not null	smallint not null	dec(5,2)	date	date
Desc:	Employee number	Project number	Activity number	Proportion of employee's time spent on project	Date activity starts	Date activity ends
Values:	000010	AD3100	10	.50	1982-01-01	1982-07-01
	000070	AD3110	10	1.00	1982-01-01	1983-02-01
	000230	AD3111	60	1.00	1982-01-01	1982-03-15
	000230	AD3111	60	.50	1982-03-15	1982-04-15
	000230	AD3111	70	.50	1982-03-15	1982-10-15
	000230	AD3111	80	.50	1982-04-15	1982-10-15
	000230	AD3111	180	1.00	1982-10-15	1983-01-01
	000240	AD3111	70	1.00	1982-02-15	1982-09-15
	000240	AD3111	80	1.00	1982-09-15	1983-01-01
	000250	AD3112	60	1.00	1982-01-01	1982-02-01
	000250	AD3112	60	.50	1982-02-01	1982-03-15
	000250	AD3112	60	.50	1982-12-01	1983-01-01
	000250	AD3112	60	1.00	1983-01-01	1983-02-01
	000250	AD3112	70	.50	1982-02-01	1982-03-15
	000250	AD3112	70	1.00	1982-03-15	1982-08-15
	000250	AD3112	70	.25	1982-08-15	1982-10-15
	000250	AD3112	80	.25	1982-08-15	1982-10-15
	000250	AD3112	80	.50	1982-10-15	1982-12-01
	000250	AD3112	180	.50	1982-08-15	1983-01-01
	000260	AD3113	70	.50	1982-06-15	1982-07-01
	000260	AD3113	70	1.00	1982-07-01	1983-02-01
	000260	AD3113	80	1.00	1982-01-01	1982-03-01
	000260	AD3113	80	.50	1982-03-01	1982-04-15
	000260	AD3113	180	.50	1982-03-01	1982-04-15
	000260	AD3113	180	1.00	1982-04-15	1982-06-01
	000260	AD3113	180	.50	1982-06-01	1982-07-01
	000270	AD3113	60	.50	1982-03-01	1982-04-01
	000270	AD3113	60	1.00	1982-04-01	1982-09-01
	000270	AD3113	60	.25	1982-09-01	1982-10-15
	000270	AD3113	70	.75	1982-09-01	1982-10-15
	000270	AD3113	70	1.00	1982-10-15	1983-02-01
	000270	AD3113	80	1.00	1982-01-01	1982-03-01
	000270	AD3113	80	.50	1982-03-01	1982-04-01
	000030	IF1000	10	.50	1982-06-01	1983-01-01

Sample Tables

Name:	EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
	000130	IF1000	90	1.00	1982-01-01	1982-10-01
	000130	IF1000	100	.50	1982-10-01	1983-01-01
	000140	IF1000	90	.50	1982-10-01	1983-01-01
	000030	IF2000	10	.50	1982-01-01	1983-01-01
	000140	IF2000	100	1.00	1982-01-01	1982-03-01
	000140	IF2000	100	.50	1982-03-01	1982-07-01
	000140	IF2000	110	.50	1982-03-01	1982-07-01
	000140	IF2000	110	.50	1982-10-01	1983-01-01
	000010	MA2100	10	.50	1982-01-01	1982-11-01
	000110	MA2100	20	1.00	1982-01-01	1982-03-01
	000010	MA2110	10	1.00	1982-01-01	1983-02-01
	000200	MA2111	50	1.00	1982-01-01	1982-06-15
	000200	MA2111	60	1.00	1982-06-15	1983-02-01
	000220	MA2111	40	1.00	1982-01-01	1983-02-01
	000150	MA2112	60	1.00	1982-01-01	1982-07-15
	000150	MA2112	180	1.00	1982-07-15	1983-02-01
	000170	MA2112	60	1.00	1982-01-01	1983-06-01
	000170	MA2112	70	1.00	1982-06-01	1983-02-01
	000190	MA2112	70	1.00	1982-02-01	1982-10-01
	000190	MA2112	80	1.00	1982-10-01	1983-10-01
	000160	MA2113	60	1.00	1982-07-15	1983-02-01
	000170	MA2113	80	1.00	1982-01-01	1983-02-01
	000180	MA2113	70	1.00	1982-04-01	1982-06-15
	000210	MA2113	80	.50	1982-10-01	1983-02-01
	000210	MA2113	180	.50	1982-10-01	1983-02-01
	000050	OP1000	10	.25	1982-01-01	1983-02-01
	000090	OP1010	10	1.00	1982-01-01	1983-02-01
	000280	OP1010	130	1.00	1982-01-01	1983-02-01
	000290	OP1010	130	1.00	1982-01-01	1983-02-01
	000300	OP1010	130	1.00	1982-01-01	1983-02-01
	000310	OP1010	130	1.00	1982-01-01	1983-02-01
	000050	OP2010	10	.75	1982-01-01	1983-02-01
	000100	OP2010	10	1.00	1982-01-01	1983-02-01
	000320	OP2011	140	.75	1982-01-01	1983-02-01
	000320	OP2011	150	.25	1982-01-01	1983-02-01
	000330	OP2012	140	.25	1982-01-01	1983-02-01
	000330	OP2012	160	.75	1982-01-01	1983-02-01
	000340	OP2013	140	.50	1982-01-01	1983-02-01
	000340	OP2013	170	.50	1982-01-01	1983-02-01
	000020	PL2100	30	1.00	1982-01-01	1982-09-15

EMP_PHOTO Table

Name:	EMPNO	PHOTO_FORMAT	PICTURE
Type:	char(6) not null	varchar(10) not null	blob(100k)
Desc:	Employee number	Photo format	Photo of employee
Values:	000130	bitmap	db200130.bmp
	000130	gif	db200130.gif
	000130	xwd	db200130.xwd
	000140	bitmap	db200140.bmp
	000140	gif	db200140.gif
	000140	xwd	db200140.xwd
	000150	bitmap	db200150.bmp
	000150	gif	db200150.gif
	000150	xwd	db200150.xwd
	000190	bitmap	db200190.bmp
	000190	gif	db200190.gif
	000190	xwd	db200190.xwd

- “Quintana Photo” on page 691 shows the picture of the employee, Delores Quintana.
- “Nicholls Photo” on page 692 shows the picture of the employee, Heather Nicholls.
- “Adamson Photo” on page 693 shows the picture of the employee, Bruce Adamson.
- “Walker Photo” on page 695 shows the picture of the employee, James Walker.

EMP_RESUME Table

Name:	EMPNO	RESUME_FORMAT	RESUME
Type:	char(6) not null	varchar(10) not null	clob(5k)
Desc:	Employee number	Resume Format	Resume of employee
Values:	000130	ascii	db200130.asc
	000130	script	db200130.scr
	000140	ascii	db200140.asc
	000140	script	db200140.scr
	000150	ascii	db200150.asc
	000150	script	db200150.scr
	000190	ascii	db200190.asc
	000190	script	db200190.scr

- “Quintana Resume” on page 691 shows the resume of the employee, Delores Quintana.

Sample Tables

- "Nicholls Resume" on page 692 shows the resume of the employee, Heather Nicholls.
- "Adamson Resume" on page 694 shows the resume of the employee, Bruce Adamson.
- "Walker Resume" on page 695 shows the resume of the employee, James Walker.

IN_TRAY Table

Name:	RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
Type:	timestamp	char(8)	char(64)	varchar(3000)
Desc:	Date and Time received	User id of person sending note	Brief description	The note

ORG Table

Name:	DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
Type:	smallint not null	varchar(14)	smallint	varchar(10)	varchar(13)
Desc:	Department number	Department name	Manager number	Division of corporation	City
Values:	10	Head Office	160	Corporate	New York
	15	New England	50	Eastern	Boston
	20	Mid Atlantic	10	Eastern	Washington
	38	South Atlantic	30	Eastern	Atlanta
	42	Great Lakes	100	Midwest	Chicago
	51	Plains	140	Midwest	Dallas
	66	Pacific	270	Western	San Francisco
	84	Mountain	290	Western	Denver

PROJECT Table

Name:	PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
Type:	char(6) not null	varchar(24) not null	char(3) not null	char(6) not null	dec(5,2)	date	date	char(6)
Desc:	Project number	Project name	Department responsible	Employee responsible	Estimated mean staffing	Estimated start date	Estimated end date	Major project, for a subproject
Values:	AD3100	ADMIN SERVICES	D01	000010	6.5	1982-01-01	1983-02-01	-
	AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	1982-01-01	1983-02-01	AD3100
	AD3111	PAYROLL PROGRAMMING	D21	000230	2	1982-01-01	1983-02-01	AD3110
	AD3112	PERSONNEL PROGRAMMING	D21	000250	1	1982-01-01	1983-02-01	AD3110

Name:	PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
	AD3113	ACCOUNT PROGRAMMING	D21	000270	2	1982-01-01	1983-02-01	AD3110
	IF1000	QUERY SERVICES	C01	000030	2	1982-01-01	1983-02-01	-
	IF2000	USER EDUCATION	C01	000030	1	1982-01-01	1983-02-01	-
	MA2100	WELD LINE AUTOMATION	D01	000010	12	1982-01-01	1983-02-01	-
	MA2110	W L PROGRAMMING	D11	000060	9	1982-01-01	1983-02-01	MA2100
	MA2111	W L PROGRAM DESIGN	D11	000220	2	1982-01-01	1982-12-01	MA2110
	MA2112	W L ROBOT DESIGN	D11	000150	3	1982-01-01	1982-12-01	MA2110
	MA2113	W L PROD CONT PROGS	D11	000160	3	1982-02-15	1982-12-01	MA2110
	OP1000	OPERATION SUPPORT	E01	000050	6	1982-01-01	1983-02-01	-
	OP1010	OPERATION	E11	000090	5	1982-01-01	1983-02-01	OP1000
	OP2000	GEN SYSTEMS SERVICES	E01	000050	5	1982-01-01	1983-02-01	-
	OP2010	SYSTEMS SUPPORT	E21	000100	4	1982-01-01	1983-02-01	OP2000
	OP2011	SCP SYSTEMS SUPPORT	E21	000320	1	1982-01-01	1983-02-01	OP2010
	OP2012	APPLICATIONS SUPPORT	E21	000330	1	1982-01-01	1983-02-01	OP2010
	OP2013	DB/DC SUPPORT	E21	000340	1	1982-01-01	1983-02-01	OP2010
	PL2100	WELD LINE PLANNING	B01	000020	1	1982-01-01	1982-09-15	MA2100

SALES Table

Name:	SALES_DATE	SALES_PERSON	REGION	SALES
Type:	date	varchar(15)	varchar(15)	int
Desc:	Date of sales	Employee's last name	Region of sales	Number of sales
Values:	12/31/1995	LUCCHESSI	Ontario-South	1
	12/31/1995	LEE	Ontario-South	3
	12/31/1995	LEE	Quebec	1
	12/31/1995	LEE	Manitoba	2
	12/31/1995	GOUNOT	Quebec	1
	03/29/1996	LUCCHESSI	Ontario-South	3
	03/29/1996	LUCCHESSI	Quebec	1
	03/29/1996	LEE	Ontario-South	2
	03/29/1996	LEE	Ontario-North	2

Sample Tables

Name:	SALES_DATE	SALES_PERSON	REGION	SALES
	03/29/1996	LEE	Quebec	3
	03/29/1996	LEE	Manitoba	5
	03/29/1996	GOUNOT	Ontario-South	3
	03/29/1996	GOUNOT	Quebec	1
	03/29/1996	GOUNOT	Manitoba	7
	03/30/1996	LUCCHESSI	Ontario-South	1
	03/30/1996	LUCCHESSI	Quebec	2
	03/30/1996	LUCCHESSI	Manitoba	1
	03/30/1996	LEE	Ontario-South	7
	03/30/1996	LEE	Ontario-North	3
	03/30/1996	LEE	Quebec	7
	03/30/1996	LEE	Manitoba	4
	03/30/1996	GOUNOT	Ontario-South	2
	03/30/1996	GOUNOT	Quebec	18
	03/30/1996	GOUNOT	Manitoba	1
	03/31/1996	LUCCHESSI	Manitoba	1
	03/31/1996	LEE	Ontario-South	14
	03/31/1996	LEE	Ontario-North	3
	03/31/1996	LEE	Quebec	7
	03/31/1996	LEE	Manitoba	3
	03/31/1996	GOUNOT	Ontario-South	2
	03/31/1996	GOUNOT	Quebec	1
	04/01/1996	LUCCHESSI	Ontario-South	3
	04/01/1996	LUCCHESSI	Manitoba	1
	04/01/1996	LEE	Ontario-South	8
	04/01/1996	LEE	Ontario-North	-
	04/01/1996	LEE	Quebec	8
	04/01/1996	LEE	Manitoba	9
	04/01/1996	GOUNOT	Ontario-South	3
	04/01/1996	GOUNOT	Ontario-North	1
	04/01/1996	GOUNOT	Quebec	3
	04/01/1996	GOUNOT	Manitoba	7

STAFF Table

Name:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
Type:	smallint not null	varchar(9)	smallint	char(5)	smallint	dec(7,2)	dec(7,2)
Desc:	Employee number	Employee name	Department number	Job type	Years of service	Current salary	Commission
Values:	10	Sanders	20	Mgr	7	18357.50	-
	20	Pernal	20	Sales	8	18171.25	612.45
	30	Marenghi	38	Mgr	5	17506.75	-
	40	O'Brien	38	Sales	6	18006.00	846.55
	50	Hanes	15	Mgr	10	20659.80	-

Name:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
	60	Quigley	38	Sales	-	16808.30	650.25
	70	Rothman	15	Sales	7	16502.83	1152.00
	80	James	20	Clerk	-	13504.60	128.20
	90	Koonitz	42	Sales	6	18001.75	1386.70
	100	Plotz	42	Mgr	7	18352.80	-
	110	Ngan	15	Clerk	5	12508.20	206.60
	120	Naughton	38	Clerk	-	12954.75	180.00
	130	Yamaguchi	42	Clerk	6	10505.90	75.60
	140	Fraye	51	Mgr	6	21150.00	-
	150	Williams	51	Sales	6	19456.50	637.65
	160	Molinare	10	Mgr	7	22959.20	-
	170	Kermisch	15	Clerk	4	12258.50	110.10
	180	Abrahams	38	Clerk	3	12009.75	236.50
	190	Sneider	20	Clerk	8	14252.75	126.50
	200	Scoutten	42	Clerk	-	11508.60	84.20
	210	Lu	10	Mgr	10	20010.00	-
	220	Smith	51	Sales	7	17654.50	992.80
	230	Lundquist	51	Clerk	3	13369.80	189.65
	240	Daniels	10	Mgr	5	19260.25	-
	250	Wheeler	51	Clerk	6	14460.00	513.30
	260	Jones	10	Mgr	12	21234.00	-
	270	Lea	66	Mgr	9	18555.50	-
	280	Wilson	66	Sales	9	18674.50	811.50
	290	Quill	84	Mgr	10	19818.00	-
	300	Davis	84	Sales	5	15454.50	806.10
	310	Graham	66	Sales	13	21000.00	200.30
	320	Gonzales	66	Sales	4	16858.20	844.00
	330	Burke	66	Clerk	1	10988.00	55.50
	340	Edwards	84	Sales	7	17844.00	1285.00
	350	Gafney	84	Clerk	5	13030.50	188.00

STAFFG Table

Note: STAFFG is only created for double-byte code pages.

Name:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
Type:	smallint not null	vargraphic(9)	smallint	graphic(5)	smallint	dec(9,0)	dec(9,0)
Desc:	Employee number	Employee name	Department number	Job type	Years of service	Current salary	Commission
Values:	10	Sanders	20	Mgr	7	18357.50	-
	20	Pernal	20	Sales	8	18171.25	612.45

Sample Tables

Name:	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
	30	Marenghi	38	Mgr	5	17506.75	-
	40	O'Brien	38	Sales	6	18006.00	846.55
	50	Hanes	15	Mgr	10	20659.80	-
	60	Quigley	38	Sales	-	16808.30	650.25
	70	Rothman	15	Sales	7	16502.83	1152.00
	80	James	20	Clerk	-	13504.60	128.20
	90	Koonitz	42	Sales	6	18001.75	1386.70
	100	Plotz	42	Mgr	7	18352.80	-
	110	Ngan	15	Clerk	5	12508.20	206.60
	120	Naughton	38	Clerk	-	12954.75	180.00
	130	Yamaguchi	42	Clerk	6	10505.90	75.60
	140	Fraye	51	Mgr	6	21150.00	-
	150	Williams	51	Sales	6	19456.50	637.65
	160	Molinare	10	Mgr	7	22959.20	-
	170	Kermisch	15	Clerk	4	12258.50	110.10
	180	Abrahams	38	Clerk	3	12009.75	236.50
	190	Sneider	20	Clerk	8	14252.75	126.50
	200	Scoutten	42	Clerk	-	11508.60	84.20
	210	Lu	10	Mgr	10	20010.00	-
	220	Smith	51	Sales	7	17654.50	992.80
	230	Lundquist	51	Clerk	3	13369.80	189.65
	240	Daniels	10	Mgr	5	19260.25	-
	250	Wheeler	51	Clerk	6	14460.00	513.30
	260	Jones	10	Mgr	12	21234.00	-
	270	Lea	66	Mgr	9	18555.50	-
	280	Wilson	66	Sales	9	18674.50	811.50
	290	Quill	84	Mgr	10	19818.00	-
	300	Davis	84	Sales	5	15454.50	806.10
	310	Graham	66	Sales	13	21000.00	200.30
	320	Gonzales	66	Sales	4	16858.20	844.00
	330	Burke	66	Clerk	1	10988.00	55.50
	340	Edwards	84	Sales	7	17844.00	1285.00
	350	Gafney	84	Clerk	5	13030.50	188.00

Sample Files with BLOB and CLOB Data Type

This section shows the data found in the EMP_PHOTO files (pictures of employees) and EMP_RESUME files (resumes of employees).

Quintana Photo

Figure 67. Delores M. Quintana

Quintana Resume

The following text is found in the db200130.asc and db200130.scr files.

Resume: Delores M. Quintana

Personal Information

Address:	1150 Eglinton Ave Mellonville, Idaho 83725
Phone:	(208) 555-9933
Birthdate:	September 15, 1925
Sex:	Female
Marital Status:	Married
Height:	5'2"
Weight:	120 lbs.

Department Information

Employee Number:	000130
Dept Number:	C01
Manager:	Sally Kwan
Position:	Analyst
Phone:	(208) 555-4578
Hire Date:	1971-07-28

Education

1965	Math and English, B.A. Adelphi University
1960	Dental Technician Florida Institute of Technology

Work History

Sample Tables

10/91 - present	Advisory Systems Analyst Producing documentation tools for engineering department.
12/85 - 9/91	Technical Writer Writer, text programmer, and planner.
1/79 - 11/85	COBOL Payroll Programmer Writing payroll programs for a diesel fuel company.

Interests

- Cooking
- Reading
- Sewing
- Remodeling

Nicholls Photo

Figure 68. Heather A. Nicholls

Nicholls Resume

The following text is found in the db200140.asc and db200140.scr files.

Resume: Heather A. Nicholls

Personal Information

Address:	844 Don Mills Ave Mellonville, Idaho 83734
Phone:	(208) 555-2310
Birthdate:	January 19, 1946
Sex:	Female
Marital Status:	Single
Height:	5'8"
Weight:	130 lbs.

Department Information

Employee Number:	000140
Dept Number:	C01
Manager:	Sally Kwan
Position:	Analyst
Phone:	(208) 555-1793
Hire Date:	1976-12-15

Education

1972	Computer Engineering, Ph.D. University of Washington
1969	Music and Physics, M.A. Vassar College

Work History

2/83 - present	Architect, OCR Development Designing the architecture of OCR products.
12/76 - 1/83	Text Programmer Optical character recognition (OCR) programming in PL/I.
9/72 - 11/76	Punch Card Quality Analyst Checking punch cards met quality specifications.

Interests

- Model railroading
- Interior decorating
- Embroidery
- Knitting

Adamson Photo

Figure 69. Bruce Adamson

Sample Tables

Adamson Resume

The following text is found in the db200150.asc and db200150.scr files.

Resume: Bruce Adamson

Personal Information

Address:	3600 Steeles Ave Mellonville, Idaho 83757
Phone:	(208) 555-4489
Birthdate:	May 17, 1947
Sex:	Male
Marital Status:	Married
Height:	6'0"
Weight:	175 lbs.

Department Information

Employee Number:	000150
Dept Number:	D11
Manager:	Irving Stern
Position:	Designer
Phone:	(208) 555-4510
Hire Date:	1972-02-12

Education

1971	Environmental Engineering, M.Sc. Johns Hopkins University
1968	American History, B.A. Northwestern University

Work History

8/79 - present	Neural Network Design Developing neural networks for machine intelligence products.
2/72 - 7/79	Robot Vision Development Developing rule-based systems to emulate sight.
9/71 - 1/72	Numerical Integration Specialist Helping bank systems communicate with each other.

Interests

- Racing motorcycles
- Building loudspeakers
- Assembling personal computers
- Sketching

Walker Photo

Figure 70. James H. Walker

Walker Resume

The following text is found in the db200190.asc and db200190.scr files.

Resume: James H. Walker

Personal Information

Address:	3500 Steeles Ave Mellonville, Idaho 83757
Phone:	(208) 555-7325
Birthdate:	June 25, 1952
Sex:	Male
Marital Status:	Single
Height:	5'11"
Weight:	166 lbs.

Department Information

Employee Number:	000190
Dept Number:	D11
Manager:	Irving Stern
Position:	Designer
Phone:	(208) 555-2986
Hire Date:	1974-07-26

Education

1974	Computer Studies, B.Sc. University of Massachusetts
1972	Linguistic Anthropology, B.A. University of Toronto

Sample Tables

Work History

6/87 - present	Microcode Design Optimizing algorithms for mathematical functions.
4/77 - 5/87	Printer Technical Support Installing and supporting laser printers.
9/74 - 3/77	Maintenance Programming Patching assembly language compiler for mainframes.

Interests

- Wine tasting
- Skiing
- Swimming
- Dancing

Appendix I. Catalog Views

The database manager creates and maintains two sets of system catalog views. This appendix contains a description of each system catalog view, including column names and data types. All the system catalog views are created when a database is created with the CREATE DATABASE command. The catalog views cannot be explicitly created or dropped. The system catalog views are updated during normal operation in response to SQL data definition statements, environment routines, and certain utilities. Data in the system catalog views is available through normal SQL query facilities. The system catalog views cannot be modified using normal SQL data manipulation commands with the exception of some specific updatable catalog views.

The catalog views are supported in addition to the catalog base tables from Version 1. The views are within the SYSCAT schema and SELECT privilege on all views is granted to PUBLIC by default. Application programs should be written to these views rather than the base catalog tables.[5] A second set of views formed from a subset of those within the SYSCAT schema, contain statistical information used by the optimizer. The views within the SYSSTAT schema contain some updatable columns.

The catalog views are designed to use more consistent conventions than the underlying catalog base tables. Columns have consistent names based on the type of objects that they describe:

Described Object	Column Names
Table	TABSCHEMA, TABNAME
Index	INDSCHEMA, INDNAME
View	VIEWSCHEMA, VIEWNAME
Constraint	CONSTSCHEMA, CONSTNAME
Trigger	TRIGSCHEMA, TRIGNAME
Package	PKGSCHEMA, PKGNAME
Type	TYPESCHEMA, TYPENAME, TYPEID
Function	FUNCSCHEMA, FUNCNAME, FUNCID
Column	COLNAME
Schema	SCHEMANAME
Table Space	TBSPACE
Nodegroup	NGNAME
Buffer pool	BPNAME
Event Monitor	EVMONNAME
Creation Timestamp	CREATE_TIME

5 Most existing applications using the base tables, however, will continue to run.

Updatable Catalog Views

The updatable views contain statistical information used by the optimizer. Some columns in these views may be changed to investigate the performance of hypothetical databases. An object (table, column, function, or index) will appear in the updatable catalog view for a given user only if that user created the object, holds CONTROL privilege on the object, or holds explicit DBADM privilege. These views are found in the SYSSTAT schema. They are defined on top of the SYSCAT views.

Before changing any statistics for the first time, it is advised to issue the RUNSTATS command so that all statistics will reflect the current state.

See *SQL Reference* for more information, including rules for updating catalog statistics.

"Roadmap" to Catalog Views

Description	Catalog View
authorities on database	SYSCAT.DBAUTH
Buffer pool configuration on nodegroup	SYSCAT.BUFFERPOOLS
Buffer pool size on node	SYSCAT.BUFFERPOOLNODES
check constraints	SYSCAT.CHECKS
column privileges	SYSCAT.COLAUTH
columns	SYSCAT.COLUMNS
columns referenced by check constraints	SYSCAT.COLCHECKS
columns used in keys	SYSCAT.KEYCOLUSE
constraint dependencies	SYSCAT.CONSTDEP
datatypes	SYSCAT.DATATYPES
event monitor definitions	SYSCAT.EVENTMONITORS
events currently monitored	SYSCAT.EVENTS
function parameters	SYSCAT.FUNCPARMS
index privileges	SYSCAT.INDEXAUTH
indexes	SYSCAT.INDEXES
detailed column statistics	SYSCAT.COLDIST
nodegroup definitions	SYSCAT.NODEGROUPS
nodegroup nodes	SYSCAT.NODEGROUPDEF
partitioning maps	SYSCAT.PARTITIONMAPS
package dependencies	SYSCAT.PACKAGEDEP
package privileges	SYSCAT.PACKAGEAUTH
packages	SYSCAT.PACKAGES
stored procedures	SYSCAT.PROCEDURES
procedure parameters	SYSCAT.PROCPARMS

Description	Catalog View
referential constraints	SYSCAT.REFERENCES
schema privileges	SYSCAT.SCHEMAAUTH
schemas	SYSCAT.SCHEMATA
statements in packages	SYSCAT.STATEMENTS
table constraints	SYSCAT.TABCONST
table privileges	SYSCAT.TABAUTH
tables	SYSCAT.TABLES
table spaces	SYSCAT.TABLESPACES
trigger dependencies	SYSCAT.TRIGDEP
triggers	SYSCAT.TRIGGERS
user-defined functions	SYSCAT.FUNCTIONS
view dependencies	SYSCAT.VIEWDEP
views	SYSCAT.TABLES SYSCAT.VIEWS

"Roadmap" to Updatable Catalog Views

Description	Catalog View
columns	SYSSTAT.COLUMNS
indexes	SYSSTAT.INDEXES
detailed column statistics	SYSSTAT.COLDIST
tables	SYSSTAT.TABLES
user-defined functions	SYSSTAT.FUNCTIONS

SYSCAT.BUFFERPOOLS

Contains a row for every buffer pool in every nodegroup.

Table 44. SYSCAT.BUFFERPOOLS Catalog View

Column Name	Data Type	Nullable	Description
BPNAME	VARCHAR(18)		Name of buffer pool
BUFFERPOOLID	INTEGER		Internal buffer pool identifier
NGNAME	VARCHAR(18)	Yes	Nodegroup name (NULL if the buffer pool exists on all nodes in the database)
NPAGES	INTEGER		Number of pages in the buffer pool
PAGESIZE	INTEGER		Pagesize for this buffer pool
ESTORE	CHAR(1)		N=This buffer pool does not use extended storage Y=This buffer pool uses extended storage

SYSCAT.BUFFERPOOLNODES

Contains a row for each node in the buffer pool for which the size of the buffer pool on the node is different from the default size in SYSCAT.BUFFERPOOLS column NPAGES.

Table 45. SYSCAT.BUFFERPOOLNODES Catalog View

Column Name	Data Type	Nullable	Description
BUFFERPOOLID	INTEGER		Internal buffer pool identifier
NODENUM	SMALLINT		Node Number
NPAGES	INTEGER		Number of pages in this buffer pool on this node

SYSCAT.CHECKS

Contains one row for each CHECK constraint.

Table 46. SYSCAT.CHECKS Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(18)		Name of the check constraint (unique within a table.)
DEFINER	CHAR(8)		Authorization ID under which the check constraint was defined.
TABSCHEMA	CHAR(8)		Qualified name of the table to which this constraint applies.
TABNAME	VARCHAR(18)		
CREATE_TIME	TIMESTAMP		The time at which the constraint was defined. Used in resolving functions that are used in this constraint. No functions will be chosen that were created after the definition of the constraint.
FUNC_PATH	VARCHAR(254)		The current function path that was used when the constraint was created.
TEXT	CLOB(32K)		The text of the CHECK clause.

SYSCAT.COLAUTH

Contains one or more rows for each user or group who is granted a column level privilege, indicating the type of privilege and whether or not it is grantable.

Table 47. SYSCAT.COLAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	CHAR(8)		Authorization ID of the user who granted the privileges or SYSIBM.
GRANTEE	CHAR(8)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U=Grantee is an individual user G=Grantee is a group
TABSCHEMA	CHAR(8)		Qualified name of the table or view.
TABNAME	VARCHAR(18)		
COLNAME	VARCHAR(18)		Name of the column to which this privilege applies.
COLNO	SMALLINT		Number of this column in the table or view.
PRIVTYPE	CHAR(1)		Indicates the type of privilege held on the table or view: U=update privilege. R=reference privilege.
GRANTABLE	CHAR(1)		Indicates if the privilege is grantable. G=grantable. N=not grantable.

SYSCAT.COLCHECKS

Each row represents some column that is referenced by a CHECK constraint.

Table 48. SYSCAT.COLCHECKS Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(18)		Name of the check constraint. (Unique within a table. May be system generated.)
TABSCHEMA	CHAR(8)		Qualified name of table containing referenced column.
TABNAME	VARCHAR(18)		
COLNAME	VARCHAR(18)		Name of column.

SYSCAT.COLDIST

Contains detailed column statistics for use by the optimizer. Each row describes the Nth-most-frequent value of some column.

Table 49. SYSCAT.COLDIST Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	CHAR(8)		Qualified name of the table to which this entry applies.
TABNAME	VARCHAR(18)		
COLNAME	VARCHAR(18)		Name of the column to which this entry applies.
TYPE	CHAR(1)		F=Frequency (most frequent value) Q=Quantile value
SEQNO	SMALLINT		If TYPE=F, then N in this column identifies the Nth most frequent value. If TYPE=Q, then N in this column identifies the Nth quantile value.
COLVALUE	VARCHAR(33)	Yes	The data value, as a character literal or a null value.
VALCOUNT	INTEGER		If TYPE=F, then VALCOUNT is the number of occurrences of COLVALUE in the column. If TYPE=Q, then VALCOUNT is the number of rows whose value is less than or equal to COLVALUE.
DISTCOUNT	INTEGER	Yes	If TYPE=Q, this column records the number of distinct values that are less than or equal to COLVALUE (null if unavailable).

SYSCAT.COLUMNS

Contains one row for each column that is defined for a table or view. All of the catalog views have entries in the SYSCAT.COLUMNS table.

Table 50 (Page 1 of 2). SYSCAT.COLUMNS Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	CHAR(8)		Qualified name of the table or view that contains the column.
TABNAME	VARCHAR(18)		
COLNAME	VARCHAR(18)		Column name.
COLNO	SMALLINT		Numerical place of column in table or view, beginning at zero.
TYPESCHEMA	CHAR(8)		Contains the qualified name of the type, if the data type of the column is distinct. Otherwise TYPESCHEMA contains the value SYSIBM and TYPENAME contains the data type of the column (in long form, for example, CHARACTER). If FLOAT or FLOAT(*n*) with *n* greater than 24 is specified, TYPENAME is renamed to DOUBLE. If FLOAT(*n*) with *n* less than 25 is specified, TYPENAME is renamed to REAL. Also, NUMERIC is renamed to DECIMAL.
TYPENAME	VARCHAR(18)		
LENGTH	INTEGER		Maximum length of data. 0 for distinct types. The LENGTH column indicates precision for DECIMAL fields.
SCALE	SMALLINT		Scale for DECIMAL fields; 0 if not DECIMAL.
DEFAULT	VARCHAR(254)	Yes	Default value for the column of a table expressed as a constant, special register, or cast-function appropriate for the data type of the column. May also be the keyword NULL. Values may be converted from what was specified as a default value. For example, date and time constants are presented in ISO format and cast-function names are qualified with schema name and the identifiers are delimited (see Note 3). Null value if a DEFAULT clause was not specified or the column is a view column.
NULLS	CHAR(1)		Y=Column is nullable. N=Column is not nullable. The value can be N for a view column that is derived from an expression or function. Nevertheless, such a column allows nulls when the statement using the view is processed with warnings for arithmetic errors. See Note 1.

Table 50 (Page 2 of 2). SYSCAT.COLUMNS Catalog View

Column Name	Data Type	Nullable	Description
CODEPAGE	SMALLINT		Code page of the column. For character-string columns not defined with the FOR BIT DATA attribute, the value is the database code page. For graphic-string columns, the value is the DBCS code page implied by the (composite) database code page. Otherwise, the value is 0.
LOGGED	CHAR(1)		Applies only to columns whose type is LOB or distinct based on LOB (blank otherwise). Y=Column is logged. N=Column is not logged.
COMPACT	CHAR(1)		Applies only to columns whose type is LOB or distinct based on LOB (blank otherwise). Y=Column is compacted in storage. N=Column is not compacted.
COLCARD	INTEGER		Number of distinct values in the column; –1 if statistics are not gathered.
HIGH2KEY	VARCHAR(33)		Second highest value of the column. This field is empty if statistics are not gathered. See Note 2.
LOW2KEY	VARCHAR(33)		Second lowest value of the column. Empty if statistics not gathered. See Note 2.
AVGCOLLEN	INTEGER		Average column length. -1 if a long field or LOB, or statistics have not been collected.
KEYSEQ	SMALLINT	Yes	The column's numerical position within the table's primary key. This field is null or 0 if the column is not part of the primary key.
PARTKEYSEQ	SMALLINT	Yes	The column's numerical position within the table's partitioning key. This field is null or 0 if the column is not part of the partitioning key.
NQUANTILES	SMALLINT		Number of quantile values recorded in SYSCAT.SYSCOLDIST for this column; -1 if no statistics.
NMOSTFREQ	SMALLINT		Number of most-frequent values recorded in SYSCAT.COLDIST for this column; -1 if statistics not gathered.
REMARKS	VARCHAR(254)	Yes	User-supplied comment.

Note:

1. Starting with Version 2, value D (indicating not null with a default) is no longer used. Instead, use of WITH DEFAULT is indicated by a non-null value in the DEFAULT column.
2. Starting with Version 2, representation of numeric data has been changed to character literals. The size has been enlarged from 16 to 33 bytes.
3. For Version 2.1.0, cast-function names were not delimited and may still appear this way in the DEFAULT column. Also, some view columns included default values which will still appear in the DEFAULT column.

SYSCAT.CONSTDEP

Contains a row for every dependency of a constraint on some other object.

Table 51. SYSCAT.CONSTDEP Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(18)		Name of the constraint.
TABSCHEMA	CHAR(8)		Qualified name of the table to which the constraint applies.
TABNAME	VARCHAR(18)		
BTYPE	CHAR(1)		Type of object that the constraint depends on. Possible values: F=function instance. I=index instance.
BSCHEMA	CHAR(8)		Qualified name of object that the constraint depends on.
BNAME	VARCHAR(18)		

SYSCAT.DATATYPES

Contains a row for every data type, including built-in and user-defined types.

Table 52. SYSCAT.DATATYPES Catalog View

Column Name	Data Type	Nullable	Description
TYPESCHEMA	CHAR(8)		Qualified name of the data type (for built-in types, TYPESCHEMA is SYSIBM).
TYPENAME	VARCHAR(18)		
DEFINER	CHAR(8)		Authorization ID under which type was created.
SOURCESCHEMA	CHAR(8)	Yes	Qualified name of the source type for distinct types. Null for other types.
SOURCENAME	VARCHAR(18)	Yes	
METATYPE	CHAR(1)		S=System predefined type T=Distinct type
TYPEID	SMALLINT		Internal type ID.
SOURCETYPEID	SMALLINT	Yes	Internal type ID of source type (null for built-in types).
LENGTH	INTEGER		Maximum length of the type. 0 for system predefined parameterized types (for example, DECIMAL and VARCHAR).
SCALE	SMALLINT		Scale for distinct types based on the system predefined DECIMAL type. 0 for all other types (including DECIMAL itself).
CODEPAGE	SMALLINT		Code page for character and graphic distinct types; 0 otherwise.
CREATE_TIME	TIMESTAMP		Creation time of the data type.
REMARKS	VARCHAR(254)	Yes	User-supplied comment, or null.

SYSCAT.DBAUTH

Records the database authorities held by users.

Table 53. SYSCAT.DBAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	CHAR(8)		SYSIBM or authorization ID of the user who granted the privileges.
GRANTEE	CHAR(8)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U=Grantee is an individual user G=Grantee is a group
DBADMAUTH	CHAR(1)		Whether grantee holds DBADM authority over the database: Y=Authority is held N=Authority is not held
CREATETABAUTH	CHAR(1)		Whether grantee can create tables in the database (CREATETAB): Y=Privilege is held N=Privilege is not held
BINDADDAUTH	CHAR(1)		Whether grantee can create new packages in the database (BINDADD): Y=Privilege is held N=Privilege is not held
CONNECTAUTH	CHAR(1)		Whether grantee can connect to the database (CONNECT): Y=Privilege is held N=Privilege is not held
NOFENCEAUTH	CHAR(1)		Whether grantee holds privilege to create non-fenced functions. Y=Privilege is held N=Privilege is not held
IMPLSCHEMAAUTH	CHAR(1)		Whether grantee can implicitly create schemas in the database (IMPLICIT_SCHEMA): Y=Privilege is held N=Privilege is not held

SYSCAT.EVENTMONITORS

Contains a row for every event monitor that has been defined.

Table 54. SYSCAT.EVENTMONITORS Catalog View

Column Name	Data Type	Nullable	Description
EVMONNAME	VARCHAR(18)		Name of event monitor.
DEFINER	CHAR(8)		Authorization ID of definer of event monitor.
TARGET_TYPE	CHAR(1)		The type of the target to which event data is written. Values: F=File P=Pipe
TARGET	VARCHAR(246)		Name of the target to which event data is written. Absolute pathname of file, or absolute name of pipe.
MAXFILES	INTEGER	Yes	Maximum number of event files that this event monitor permits in an event path. Null if there is no maximum, or if the target-type is not FILE.
MAXFILESIZE	INTEGER	Yes	Maximum size (in 4K pages) that each event file can reach before the event monitor creates a new file. Null if there is no maximum, or if the target-type is not FILE.
BUFFERSIZE	INTEGER	Yes	Size of buffers (in 4K pages) used by event monitors with file targets; otherwise null.
IO_MODE	CHAR(1)	Yes	Mode of file I/O. B=Blocked N=Not blocked. Null if target-type is not FILE.
WRITE_MODE	CHAR(1)	Yes	Indicates how this monitor handles existing event data when the monitor is activated. Values: A=Append R=Replace Null if target-type is not FILE.
AUTOSTART	CHAR(1)		The event monitor will be activated automatically when the database starts. Y=Yes N=No
NODENUM	SMALLINT		The number of the partition (or node) on which the event monitor runs and logs events
MONSCOPE	CHAR(1)		Monitoring scope: L=Local G=Global
REMARKS	VARCHAR(254)	Yes	Reserved for future use.

SYSCAT.EVENTS

Contains a row for every event that is being monitored. An event monitor, in general, monitors multiple events.

Table 55. SYSCAT.EVENTS Catalog View

Column Name	Data Type	Nullable	Description
EVMONNAME	VARCHAR(18)		Name of event monitor that is monitoring this event.
TYPE	VARCHAR(18)		Type of event being monitored. Possible values: DATABASE CONNECTIONS TABLES STATEMENTS TRANSACTIONS DEADLOCKS TABLESPACES
FILTER	CLOB(32K)	Yes	The full text of the WHERE-clause that applies to this event.

SYSCAT.FUNCPARMS

Contains a row for every parameter or result of a function defined in SYSCAT.FUNCTIONS.

Table 56. SYSCAT.FUNCPARMS Catalog View

Column Name	Data Type	Nullable	Description
FUNCSCHEMA	CHAR(8)		Qualified function name.
FUNCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		The name of the function instance (may be system-generated).
ROWTYPE	CHAR(1)		P=parameter R=result before casting C=result after casting
ORDINAL	SMALLINT		If ROWTYPE=P, the parameter's numerical position within the function signature. Otherwise 0.
PARMNAME	VARCHAR(18)		Name of parameter or result column, or null if no name exists.
TYPESCHEMA	CHAR(8)		Qualified name of data type of parameter or result.
TYPENAME	VARCHAR(18)		
LENGTH	INTEGER		Length of parameter or result. 0 if parameter or result is a distinct type. See Note 1.
SCALE	SMALLINT		Scale of parameter or result. 0 if parameter or result is a distinct type. See Note 1.
CODEPAGE	SMALLINT		Code page of parameter. 0 denotes either not applicable or a column for character data declared with the FOR BIT DATA attribute.
CAST_FUNCID	INTEGER	Yes	Internal function ID.
AS_LOCATOR	CHAR(1)		Y=Parameter or result is passed in the form of a locator N=Not passed in the form of a locator.

Note:

1. LENGTH and SCALE are set to 0 for sourced functions (functions defined with a reference to another function) because they inherit the length and scale of parameters from their source.

SYSCAT.FUNCTIONS

Contains a row for each user-defined function (scalar, table or sourced). Does not include built-in functions.

Table 57 (Page 1 of 3). SYSCAT.FUNCTIONS Catalog View

Column Name	Data Type	Nullable	Description
FUNCSCHEMA	CHAR(8)		Qualified function name.
FUNCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		The name of the function instance (may be system-generated).
DEFINER	CHAR(8)		Authorization ID of function definer.
FUNCID	INTEGER		Internally-assigned function ID.
RETURN_TYPE	SMALLINT		Internal type code of return type of function.
ORIGIN	CHAR(1)		B=Built-in E=User-defined, external U=User-defined, based on a source S=System-generated
TYPE	CHAR(1)		S=Scalar function C=Column function T=Table function
PARM_COUNT	SMALLINT		Number of function parameters.
PARM_SIGNATURE	VARCHAR(180) FOR BIT DATA		Concatenation of up to 90 parameter types, in internal format. Zero length if function takes no parameters.
CREATE_TIME	TIMESTAMP		Timestamp of function creation. Set to 0 for Version 1 functions.
VARIANT	CHAR(1)		Y=Variant (results may differ) N=Invariant (results are consistent) Blank if ORIGIN is not E
SIDE_EFFECTS	CHAR(1)		E=Function has external side-effects (number of invocations is important) N=No side-effects Blank if ORIGIN is not E
FENCED	CHAR(1)		Y=Fenced N=Not fenced Blank if ORIGIN is not E
NULLCALL	CHAR(1)		Y=Nullcall N=No nullcall (function result is implicitly null if operand(s) are null). Blank if ORIGIN is not E.
CAST_FUNCTION	CHAR(1)		Y=This is a cast function N=This is not a cast function
ASSIGN_FUNCTION	CHAR(1)		Y=Implicit assignment function N=Not an assignment function

Table 57 (Page 2 of 3). SYSCAT.FUNCTIONS Catalog View

Column Name	Data Type	Nullable	Description
SCRATCHPAD	CHAR(1)		Y=This function has a scratch pad N=This function does not have a scratch pad Blank if ORIGIN is not E
FINAL_CALL	CHAR(1)		Y=Final call is made to this function at run time end-of-statement. N=No final call is made. Blank if ORIGIN is not E
PARALLELIZABLE	CHAR(1)		Y=Function can be executed in parallel N=Function cannot be executed in parallel Blank if ORIGIN is not E
CONTAINS_SQL	CHAR(1)		Indicates wheter an external function contains SQL. N=Function does not contain SQL statements. R=Contains read-only SQL statements. M=Contains SQL statements that modify data. Blank if ORIGIN is not E
DBINFO	CHAR(1)		Indicates whether a DBINFO parameter is passed to an external function. Y=DBINFO is passed. N=DBINFO is not passed. Blank if ORIGIN is not E
RESULT_COLS	SMALLINT		For a table function (TYPE=T) contains the number of columns in the result table; otherwise contains 1.
LANGUAGE	CHAR(8)		Implementation language of function body. Possible values are C, JAVA or OLE. Blank if ORIGIN is not E.
IMPLEMENTATION	VARCHAR(254)	Yes	If ORIGIN=E, identifies the path/module/function that implements this function. If ORIGIN=U and the source function is built-in, this column contains the name and signature of the source function. Null otherwise.
PARM_STYLE	CHAR(8)		Indicates the parameter style declared in the CREATE FUNCTION statement. Values: DB2SQL DB2GENRL
SOURCE_SCHEMA	CHAR(8)	Yes	If ORIGIN=U and the source function is a user-defined function, contains the qualified name of the source function. If ORIGIN=U and the source function is built-in, SOURCE_SCHEMA is 'SYSIBM' and SOURCE_SPECIFIC is 'N/A for built-in'. Null if ORIGIN is not U.
SOURCE_SPECIFIC	VARCHAR(18)	Yes	
IOS_PER_INVOC	DOUBLE		Estimated number of I/Os per invocation; -1 if not known (0 default).

SYSCAT.FUNCTIONS

Table 57 (Page 3 of 3). SYSCAT.FUNCTIONS Catalog View

Column Name	Data Type	Nullable	Description
INSTS_PER_INVOC	DOUBLE		Estimated number of instructions per invocation; -1 if not known (450 default).
IOS_PER_ARGBYTE	DOUBLE		Estimated number of I/O's per input argument byte; -1 if not known (0 default).
INSTS_PER_ARGBYTE	DOUBLE		Estimated number of instructions per input argument byte; -1 if not known (0 default).
PERCENT_ARGBYTES	SMALLINT		Estimated average percent of input argument bytes that the function will actually read; -1 if not known (100 default).
INITIAL_IOS	DOUBLE		Estimated number of I/O's performed the first/last time the function is invoked; -1 if not known (0 default).
INITIAL_INSTS	DOUBLE		Estimated number of instructions executed the first/last time the function is invoked; -1 if not known (0 default).
CARDINALITY	INTEGER	Yes	The predicted cardinality of a table function. –1 if not known or if function is not a table function.
REMARKS	VARCHAR(254)	Yes	User-supplied comment, or null.

SYSCAT.INDEXAUTH

Contains a row for every privilege held on an index.

Table 58. SYSCAT.INDEXAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	CHAR(8)		Authorization ID of the user who granted the privileges.
GRANTEE	CHAR(8)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U=Grantee is an individual user G=Grantee is a group
INDSCHEMA	CHAR(8)		Name of the index.
INDNAME	VARCHAR(18)		
CONTROLAUTH	CHAR(1)		Whether grantee holds CONTROL privilege over the index: Y=Privilege is held N=Privilege is not held

SYSCAT.INDEXES

Contains one row for each index that is defined for a table.

Table 59 (Page 1 of 2). SYSCAT.INDEXES Catalog View

Column Name	Data Type	Nullable	Description
INDSCHEMA	CHAR(8)		Name of the index.
INDNAME	VARCHAR(18)		
DEFINER	CHAR(8)		User who created the index.
TABSCHEMA	CHAR(8)		Qualified name of the table on which the index is defined.
TABNAME	VARCHAR(18)		
COLNAMES	VARCHAR(320)		List of column names, each preceded by + or – to indicate ascending or descending order respectively.
UNIQUERULE	CHAR(1)		Unique rule: D=duplicates allowed U=unique entries only allowed P=primary index.
MADE_UNIQUE	CHAR(1)		Y=Index was originally non-unique but was converted to a unique index to support a unique or primary key constraint. If the constraint is dropped, the index will revert to non-unique. N=Index remains as it was created.
COLCOUNT	SMALLINT		Number of columns in the key.
UNIQUE_COLCOUNT	SMALLINT		The number of columns required for a unique key. Always <=COLCOUNT. –1 if index has no unique key (permits duplicates)
INDEXTYPE	CHAR(4)		Type of index. REG =Regular
PCTFREE	SMALLINT		Percentage of each index page to be reserved during initial building of the index. This space is available for future inserts after the index is built.
IID	SMALLINT		Internal index ID.
NLEAF	INTEGER		Number of leaf pages; –1 if statistics are not gathered.
NLEVELS	SMALLINT		Number of index levels; –1 if statistics are not gathered.
FIRSTKEYCARD	INTEGER		Number of distinct first key values; –1 if statistics are not gathered.
FIRST2KEYCARD	INTEGER		Number of distinct keys using the first two columns of the index (–1 if no statistics or inapplicable)

Table 59 (Page 2 of 2). SYSCAT.INDEXES Catalog View

Column Name	Data Type	Nullable	Description
FIRST3KEYCARD	INTEGER		Number of distinct keys using the first three columns of the index (–1 if no statistics or inapplicable)
FIRST4KEYCARD	INTEGER		Number of distinct keys using the first four columns of the index (–1 if no statistics or inapplicable)
FULLKEYCARD	INTEGER		Number of distinct full key values; –1 if statistics are not gathered.
CLUSTERRATIO	SMALLINT		Degree of data clustering with the index; –1 if statistics are not gathered or if detailed index statistics are gathered (in which case, CLUSTERFACTOR will be used instead).
CLUSTERFACTOR	DOUBLE		Finer measurement of degree of clustering, or -1 if detailed index statistics have not been gathered.
SEQUENTIAL_PAGES	INTEGER		Number of leaf pages located on disk in index key order with few or no large gaps between them. (–1 if no statistics are available.)
DENSITY	INTEGER		Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percent (integer between 0 and 100, –1 if no statistics are available.)
USER_DEFINED	SMALLINT		1 if this index was defined by a user and has not been dropped; otherwise 0.
SYSTEM_REQUIRED	SMALLINT		1 if this index is required for primary key or unique key constraint; otherwise 0.
CREATE_TIME	TIMESTAMP		Time when the index was created.
STATS_TIME	TIMESTAMP	Yes	Last time when any change was made to recorded statistics for this index. Null if no statistics available.
PAGE_FETCH_PAIRS	VARCHAR(254)		A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer, and the number of page fetches required to scan the table with this index using that hypothetical buffer. (Zero-length string if no data available.)
REMARKS	VARCHAR(254)	Yes	User-supplied comment, or null.
TEXT	CLOB(32K)	Yes	Reserved for future use.

SYSCAT.KEYCOLUSE

Lists all columns that participate in a key defined by a unique, primary key, or foreign key constraint.

Table 60. SYSCAT.KEYCOLUSE Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(18)		Name of the constraint (unique within a table).
TABSCHEMA	CHAR(8)		Qualified name of the table containing the column.
TABNAME	VARCHAR(18)		
COLNAME	VARCHAR(18)		Name of the column.
COLSEQ	SMALLINT		Numeric position of the column in the key (initial position=1).

SYSCAT.NODEGROUPDEF

Contains a row for each partition that is contained in a nodegroup.

Table 61. SYSCAT.NODEGROUPDEF Catalog View

Column Name	Data Type	Nullable	Description
NGNAME	VARCHAR(18)		The name of the nodegroup that contains the partition (or node) .
NODENUM	SMALLINT		The partition (or node) number of a partition contained in the nodegroup. A valid partition number is between 0 and 999 inclusive.
IN_USE	CHAR(1)		Status of the partition (or node) . A The newly added partition is not in the partitioning map but the containers for the table spaces in the nodegroup are created. The partition is added to the partitioning map when a Redistribute Nodegroup operation is successfully completed. D The partition will be dropped when a Redistribute Nodegroup operation is completed. T The newly added partition is not in the partitioning map and it was added using the WITHOUT TABLESPACES clause. Containers must be specifically added to the table spaces for the nodegroup. Y The partition is in the partitioning map.

SYSCAT.NODEGROUPS

Contains a row for each nodegroup.

Table 62. SYSCAT.NODEGROUPS Catalog View

Column Name	Data Type	Nullable	Description
NGNAME	VARCHAR(18)		Name of the nodegroup.
DEFINER	CHAR(8)		Authorization ID of the nodegroup definer.
PMAP_ID	SMALLINT		Identifier of the partitioning map in SYSCAT.PARTITIONMAPS.
REBALANCE_PMAP_ID	SMALLINT		Identifier of the partitioning map currently being used for re-distribution. Value is -1 if re-distribution is currently not in progress.
CREATE_TIME	TIMESTAMP		Creation time of nodegroup.
REMARKS	VARCHAR(254)	Yes	User-provided comment.

SYSCAT.PACKAGEAUTH

Contains a row for every privilege held on a package.

Table 63. SYSCAT.PACKAGEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	CHAR(8)		Authorization ID of the user who granted the privileges.
GRANTEE	CHAR(8)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U=Grantee is an individual user G=Grantee is a group
PKGSCHEMA	CHAR(8)		Name of the package on which the privileges are held.
PKGNAME	CHAR(8)		
CONTROLAUTH	CHAR(1)		Indicates whether grantee holds CONTROL privilege on the package: Y=Privilege is held. N=Privilege is not held.
BINDAUTH	CHAR(1)		Indicates whether grantee holds BIND privilege on the package: Y=Privilege is held. N=Privilege is not held.
EXECUTEAUTH	CHAR(1)		Indicates whether grantee holds EXECUTE privilege on the package: Y=Privilege is held. N=Privilege is not held.

SYSCAT.PACKAGEDEP

Contains a row for each dependency that packages have on indexes, tables, views, functions, and aliases.

Table 64. SYSCAT.PACKAGEDEP Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	CHAR(8)		Name of the package.
PKGNAME	CHAR(8)		
BINDER	CHAR(8)	Yes	Binder of the package.
BTYPE	CHAR(1)		Type of object BNAME: A=alias F=function-instance I=index T=table V=view
BSCHEMA	CHAR(8)		Qualified name of an object on which the package is dependent.
BNAME	VARCHAR(18)		
TABAUTH	SMALLINT	Yes	If BTYPE is T(table) or V(view), encodes the privileges that are required by this package (Select, Insert, Delete, Update).

Note:

1. When a depended-on function-instance is dropped, the package is placed into an “inoperative” state from which it must be explicitly rebound. When any other depended-on object is dropped, the package is placed into an “invalid” state from which the system will attempt to rebind it automatically when a package is first referenced.

SYSCAT.PACKAGES

Contains a row for each package that has been created by binding an application program.

Table 65 (Page 1 of 3). SYSCAT.PACKAGES Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	CHAR(8)		Name of the package.
PKGNAME	CHAR(8)		
BOUNDBY	CHAR(8)		Authorization ID of the binder of the package.
DEFINER	CHAR(8)		Userid under which package was bound.
DEFAULT_SCHEMA	CHAR(8)		Default schema name used for unqualified names in static SQL statements.
VALID	CHAR(1)		Y=Valid N=Not valid X=Package is inoperative because some function instance that it depends on has been dropped. Explicit rebind is needed. See Note 1 on "SYSCAT.PACKAGEDEP" on page 724
UNIQUE_ID	CHAR(8)		Internal date and time information indicating when the package was first created.
TOTAL_SECT	SMALLINT		Total number of sections in the package.
FORMAT	CHAR(1)		Date and time format associated with the package: 0=Format associated with country code of the database 1=USA date and time 2=EUR date, EUR time 3=ISO date, ISO time. 4=JIS date, JIS time. 5=LOCAL date, LOCAL time.
ISOLATION	CHAR(2)	Yes	Isolation level: RR=Repeatable read RS=Read stability CS=Cursor stability UR=Uncommitted read.
BLOCKING	CHAR(1)	Yes	Cursor blocking option: N=No blocking U=Block unambiguous cursors B=Block all cursors
INSERT_BUF	CHAR(1)		Insert option used during bind: Y=Inserts are buffered N=Inserts are not buffered

SYSCAT.PACKAGES

Table 65 (Page 2 of 3). SYSCAT.PACKAGES Catalog View

Column Name	Data Type	Nullable	Description
LANG_LEVEL	CHAR(1)	Yes	LANGLEVEL value used during BIND: 0=SAA1 1=SQL92E or MIA
FUNC_PATH	VARCHAR(254)		The function path used by the last BIND command for this package. This is used as the default path for REBIND. SYSIBM for pre-Version 2 packages.
QUERYOPT	INTEGER		Optimization class under which this package was bound. Used for rebind. The classes are: 0, 1, 3, 5 and 9. .
EXPLAIN_LEVEL	CHAR(1)		Indicates whether Explain was requested using the EXPLAIN or EXPLSNAP bind option. Blank=No Explain requested P=Plan Selection level
EXPLAIN_MODE	CHAR(1)		Value of EXPLAIN bind option: Y=Yes (static) N=No A=All (static and dynamic)
EXPLAIN_SNAPSHOT	CHAR(1)		Value of EXPLSNAP bind option: Y=Yes (static) N=No A=All (static and dynamic)
SQLWARN	CHAR(1)		Are positive SQLCODES resulting from dynamic SQL statements returned to the application? Y=Yes N=No, they are suppressed
SQLMATHWARN	CHAR(1)		Value of database configuration parameter DFT_SQLMATHWARN at time of bind. Are arithmetic errors and retrieval conversion errors in static SQL statements handled as nulls with a warning? Y=Yes N=No, they are suppressed
EXPLICIT_BIND_TIME	TIMESTAMP		The time at which this package was last explicitly bound or rebound. When the package is implicitly rebound, no function instance will be selected that was created later than this time.
LAST_BIND_TIME	TIMESTAMP		Time at which the package last explicitly or implicitly bound or rebound.
CODEPAGE	SMALLINT		Application codepage at bind time (-1 if not known).

Table 65 (Page 3 of 3). SYSCAT.PACKAGES Catalog View

Column Name	Data Type	Nullable	Description
DEGREE	CHAR(5)		Indicates the limit on intra-partition parallelism (as a bind option) when package was bound. 1 = No intra-partition parallelism. 2 - 32767 = Degree of intra-partition parallelism. ANY = Degree was determined by the database manager.
MULTINODE_PLANS	CHAR(1)		Y =Package was bound in a multiple partition environment. N =Package was bound in a single partition environment.
INTRA_PARALLEL	CHAR(1)		Indicates the use of intra-partition parallelism by static SQL statements within the package. Y = one or more static SQL statement in package uses intra-partition parallelism. N = no static SQL statement in package uses intra-partition parallelism. F = one or more static SQL statement in package can use intra-partition parallelism; this parallelism has been disabled for use on a system that is not configured for intra-partition parallelism.
REMARKS	VARCHAR(254)	Yes	User-supplied comment, or null.

SYSCAT.PARTITIONMAPS

Contains a row for each partitioning map that is used to distribute the rows of tables among the partitions in a nodegroup, based on hashing the tables partitioning key.

Table 66. SYSCAT.PARTITIONMAPS Catalog View

Column Name	Data Type	Nullable	Description
PMAP_ID	SMALLINT		Identifier of the partitioning map.
PARTITIONMAP	LONG VARCHAR FOR BIT DATA		The actual partitioning map, a vector of 4096 two-byte integers for a multiple node nodegroup. For a single node nodegroup, there is one entry denoting the partition (or node) number of the single node.

SYSCAT.PROCEDURES

Contains a row for each stored procedure that is created.

Table 67. SYSCAT.PROCEDURES Catalog View

Column Name	Data Type	Nullable	Description
PROCSCHEMA	CHAR(8)		Qualified procedure name.
PROCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		The name of the procedure instance (may be system generated).
PROCEDURE_ID	INTEGER		Internal ID of stored procedure.
DEFINER	CHAR(8)		Authorization of the procedure definer.
PARM_COUNT	SMALLINT		Number of procedure parameters.
PARM_SIGNATURE	VARCHAR(180) FOR BIT DATA		Concatenation of up to 90 parameter types, in internal format. Zero length if procedure takes no parameters.
ORIGIN	CHAR(1)		Always 'E' = User defined, external
CREATE_TIME	TIMESTAMP		Timestamp of procedure registration.
DETERMINISTIC	CHAR(1)		Y=Results are deterministic. N=Results are not deterministic.
FENCED	CHAR(1)		Y=Fenced N=Not Fenced
NULLCALL	CHAR(1)		Always Y=NULLCALL
LANGUAGE	CHAR(8)		Implementation language of procedure body. Possible values are C and JAVA.
IMPLEMENTATION	VARCHAR(254)	Yes	Identifies the path/module/function or class/method that implements the procedure.
PARM_STYLE	CHAR(8)		DB2DARI=Language is C DB2GENRL=Language is Java
RESULT_SETS	SMALLINT		Estimated upper limit of returned result sets.
REMARKS	VARCHAR(254)	Yes	User supplied comment, or null.

SYSCAT.PROCPARMS

Contains a row for each parameter of a stored procedure.

Table 68. SYSCAT.PROCPARMS Catalog View

Column Name	Data Type	Nullable	Description
PROCSCHEMA	CHAR(8)		Qualified procedure name.
PROCNAME	VARCHAR(18)		
SPECIFICNAME	VARCHAR(18)		The name of the procedure instance (may be system generated).
ORDINAL	SMALLINT		The parameter's numerical position within the procedure signature.
PARMNAME	VARCHAR(18)		Parameter name.
TYPESCHEMA	CHAR(8)		Qualified name of data type of the parameter.
TYPENAME	VARCHAR(18)		
LENGTH	INTEGER		Length of the parameter.
SCALE	SMALLINT		Scale of the parameter.
CODEPAGE	SMALLINT		Code page of parameter. 0 denotes either not applicable or a parameter for character data declared with the FOR BIT DATA attribute.
PARM_MODE	VARCHAR(5)		IN=Input, OUT=Output, INOUT=Input/output
AS_LOCATOR	CHAR(1)		Always 'N'

SYSCAT.REFERENCES

Contains a row for each defined referential constraint.

Table 69. SYSCAT.REFERENCES Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(18)		Name of constraint.
TABSCHEMA	CHAR(8)		Qualified name of the constraint.
TABNAME	VARCHAR(18)		
DEFINER	CHAR(8)		User who created the constraint.
REFKEYNAME	VARCHAR(18)		Name of parent key.
REFTABSCHEMA	CHAR(8)		Name of the parent table.
REFTABNAME	VARCHAR(18)		
COLCOUNT	SMALLINT		Number of columns in the foreign key.
DELETERULE	CHAR(1)		Delete rule: A=NO ACTION C=CASCADE N=SET NULL R=RESTRICT
UPDATERULE	CHAR(1)		Update rule: A=NO ACTION R=RESTRICT
CREATE_TIME	TIMESTAMP		The timestamp when the referential constraint was defined.
FK_COLNAMES	VARCHAR(320)		List of foreign key column names.
PK_COLNAMES	VARCHAR(320)		List of parent key column names.

Note:

1. The SYSCAT.REFERENCES view is based on the SYSIBM.SYSRELS table from Version 1.

SYSCAT.SCHEMAAUTH

Contains one or more rows for each user or group who is granted a privilege on a particular schema in the database. All schema privileges for a single schema granted by a specific grantor to a specific grantee appear in a single row.

Table 70. SYSCAT.SCHEMAAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	CHAR(8)		Authorization ID of the user who granted the privileges or SYSIBM.
GRANTEE	CHAR(8)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U=Grantee is an individual user G=Grantee is a group
SCHEMANAME	CHAR(8)		Name of the schema.
ALTERINAUTH	CHAR(1)		Indicates whether grantee holds ALTERIN privilege on the schema: Y=Privilege is held G=Privilege is held and grantable N=Privilege is not held.
CREATEINAUTH	CHAR(1)		Indicates whether grantee holds CREATEIN privilege on the schema: Y=Privilege is held G=Privilege is held and grantable N=Privilege is not held.
DROPINAUTH	CHAR(1)		Indicates whether grantee holds DROPIN privilege on the schema: Y=Privilege is held G=Privilege is held and grantable N=Privilege is not held.

SYSCAT.SCHEMATA

Contains a row for each schema.

Table 71. SYSCAT.SCHEMATA Catalog View

Column Name	Data Type	Nullable	Description
SCHEMANAME	CHAR(8)		Name of the schema.
OWNER	CHAR(8)		Authorization id of the schema. The value for implicitly created schemas is SYSIBM.
DEFINER	CHAR(8)		User who created the schema.
CREATE_TIME	TIMESTAMP		Timstamp indicating when the object was created.
REMARKS	VARCHAR(254)	Yes	User-provided comment.

SYSCAT.STATEMENTS

Contains one or more rows for each SQL statement in each package in the database.

Table 72. SYSCAT.STATEMENTS Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	CHAR(8)		Name of the package.
PKGNAME	CHAR(8)		
STMTNO	SMALLINT		Line number of the SQL statement in the source module of the application program.
SECTNO	SMALLINT		Number of the package section containing the SQL statement.
SEQNO	SMALLINT		Sequence number of this row; the first portion of the SQL text is stored on row one, and successive rows have increasing values for SEQNO.
TEXT	VARCHAR (3600)		Text or portion of the text of the SQL statement.

SYSCAT.TABAUTH

Contains one or more rows for each user or group who is granted a privilege on a particular table or view in the database. All the table privileges for a single table or view granted by a specific grantor to a specific grantee appear in a single row.

Table 73 (Page 1 of 2). SYSCAT.TABAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	CHAR(8)		Authorization ID of the user who granted the privileges or SYSIBM.
GRANTEE	CHAR(8)		Authorization ID of the user or group who holds the privileges.
GRANTEETYPE	CHAR(1)		U=Grantee is an individual user G=Grantee is a group
TABSCHEMA	CHAR(8)		Qualified name of the table or view.
TABNAME	VARCHAR(18)		
CONTROLAUTH	CHAR(1)		Indicates whether grantee holds CONTROL privilege on the table or view: Y=Privilege is held. N=Privilege is not held.
ALTERAUTH	CHAR(1)		Indicates whether grantee holds ALTER privilege on the table: Y=Privilege is held. N=Privilege is not held. G=Privilege is held and grantable.
DELETEAUTH	CHAR(1)		Indicates whether grantee holds DELETE privilege on the table or view: Y=Privilege is held. N=Privilege is not held. G=Privilege is held and grantable.
INDEXAUTH	CHAR(1)		Indicates whether grantee holds INDEX privilege on the table: Y=Privilege is held. N=Privilege is not held. G=Privilege is held and grantable.
INSERTAUTH	CHAR(1)		Indicates whether grantee holds INSERT privilege on the table or view: Y=Privilege is held. N=Privilege is not held. G=Privilege is held and grantable.
SELECTAUTH	CHAR(1)		Indicates whether grantee holds SELECT privilege on the table or view: Y=Privilege is held. N=Privilege is not held. G=Privilege is held and grantable.

SYSCAT.TABAUTH

Table 73 (Page 2 of 2). SYSCAT.TABAUTH Catalog View

Column Name	Data Type	Nullable	Description
REFAUTH	CHAR(1)		Indicates whether grantee holds REFERENCE privilege on the table or view: Y=Privilege is held. N=Privilege is not held. G=Privilege is held and grantable.
UPDATEAUTH	CHAR(1)		Indicates whether grantee holds UPDATE privilege on the table or view: Y=Privilege is held. N=Privilege is not held. G=Privilege is held and grantable.

SYSCAT.TABCONST

Each row represents a table constraint of type CHECK, UNIQUE, PRIMARY KEY, or FOREIGN KEY.

Table 74. SYSCAT.TABCONST Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(18)		Name of the constraint (unique within a table).
TABSCHEMA	CHAR(8)		Qualified name of the table to which this constraint applies.
TABNAME	VARCHAR(18)		
DEFINER	CHAR(8)		Authorization ID under which the constraint was defined.
TYPE	CHAR(1)		Indicates the constraint type: K=CHECK P=PRIMARY KEY F=FOREIGN KEY U=UNIQUE
REMARKS	VARCHAR(254)	Yes	User-supplied comment, or null.

SYSCAT.TABLES

Contains one row for each table, view, or alias that is created. All of the catalog tables and views have entries in the SYSCAT.TABLES catalog view.

Table 75 (Page 1 of 2). SYSCAT.TABLES Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	CHAR(8)		Qualified name of the table, view, or alias.
TABNAME	VARCHAR(18)		
DEFINER	CHAR(8)		User who created the table, view, or alias.
TYPE	CHAR(1)		The type of object: A=Alias T=Table V=View
STATUS	CHAR(1)		The type of object: N=Normal table, view or alias C=Check pending on table X=Inoperative view
BASE_TABSCHEMA	CHAR(8)	Yes	If TYPE=A, these columns identify the table, view, or alias that is referenced by this alias; otherwise they are null.
BASE_TABNAME	VARCHAR(18)	Yes	
CREATE_TIME	TIMESTAMP		The timestamp indicating when the object was created.
STATS_TIME	TIMESTAMP	Yes	Last time when any change was made to recorded statistics for this table. Null if no statistics available.
COLCOUNT	SMALLINT		Number of columns in table.
TABLEID	SMALLINT		Internal table identifier.
TBSPACEID	SMALLINT		Internal identifier of primary table space for this table.
CARD	INTEGER		Total number of rows in the table; –1 if statistics are not gathered or the row describes a view or alias.
NPAGES	INTEGER		Total number of pages on which the rows of the table exist; –1 if statistics are not gathered or the row describes a view or alias.
FPAGES	INTEGER		Total number of pages; –1 if statistics are not gathered or the row describes a view or alias.
OVERFLOW	INTEGER		Total number of overflow records in the table; –1 if statistics are not gathered or the row describes a view or alias.
TBSPACE	VARCHAR(18)	Yes	Name of primary table space for the table. If no other table space is specified, all parts of the table are stored in this table space. Null for aliases and views.
INDEX_TBSPACE	VARCHAR(18)	Yes	Name of table space that holds all indexes created on this table. Null for aliases and views, or if the INDEX IN clause was omitted or specified with the same value as the IN clause of the CREATE TABLE statement.

Table 75 (Page 2 of 2). SYSCAT.TABLES Catalog View

Column Name	Data Type	Nullable	Description
LONG_TBSPACE	VARCHAR(18)	Yes	Name of table space that holds all long data (LONG or LOB column types) for this table. Null for aliases and views, or if the LONG IN clause was omitted or specified with the same value as the IN clause of the CREATE TABLE statement.
PARENTS	SMALLINT	Yes	Number of parent tables of this table (the number of referential constraints in which this table is a dependent).
CHILDREN	SMALLINT	Yes	Number of dependent tables of this table (the number of referential constraints in which this table is a parent).
SELFREFS	SMALLINT	Yes	Number of self-referencing referential constraints for this table (the number of referential constraints in which this table is both a parent and a dependent).
KEYCOLUMNS	SMALLINT	Yes	Number of columns in the primary key of the table.
KEYINDEXID	SMALLINT	Yes	Index ID of the primary index. This field is null or 0 if there is no primary key.
KEYUNIQUE	SMALLINT		Number of unique constraints (other than primary key) defined on this table.
CHECKCOUNT	SMALLINT		Number of check constraints defined on this table.
DATACAPTURE	CHAR(1)		Y=Table participates in data replication N=Does not participate
CONST_CHECKED	CHAR(32)		Byte 1 represents foreign key constraints. Byte 2 represents check constraints. Other bytes are reserved. Encodes constraint information on checking. Values: Y=Checked by system U=Checked by user N=Not checked (pending)
PMAP_ID	SMALLINT	Yes	Identifier of the partitioning map used by this table. Null for aliases and views.
PARTITION_MODE	CHAR(1)		Mode used for tables in a partitioned database. **H** hash on the partitioning key Blank for aliases, views and tables in single partition nodegroups with no partitioning key defined.
LOG_ATTRIBUTE	CHAR(1)		0=Default logging N=Table created not logged initially
PCTFREE	SMALLINT		Percentage of each page to be reserved for future inserts. Can be changed by ALTER TABLE.
REMARKS	VARCHAR(254)	Yes	User-provided comment.

SYSCAT.TABLESPACES

Contains a row for each table space.

Table 76. SYSCAT.TABLESPACES Catalog View

Column Name	Data Type	Nullable	Description
TBSPACE	VARCHAR(18)		Name of table space.
DEFINER	CHAR(8)		Authorization ID of table space definer.
CREATE_TIME	TIMESTAMP		Creation time of table space.
TBSPACEID	INTEGER		Internal table space identifier.
TBSPACETYPE	CHAR(1)		The type of the table space: S=System managed space D=Database managed space
DATATYPE	CHAR(1)		Type of data that can be stored: A=All types of permanent data L=Long data only T=Temporary tables only
EXTENTSIZE	INTEGER		Size of extent, in 4K pages. This many pages are written to one container in the table space before switching to the next container.
PREFETCHSIZE	INTEGER		Number of 4K pages to be read when prefetch is performed.
OVERHEAD	DOUBLE		Controller overhead and disk seek and latency time in milliseconds.
TRANSFERRATE	DOUBLE		Time to read one 4K page into the buffer.
PAGESIZE	INTEGER		Size (in bytes) of pages in the table space.
NGNAME	VARCHAR(18)		Name of the nodegroup for the table space.
BUFFERPOOLID	INTEGER		ID of buffer pool used by this tablespace (1 indicates default buffer pool).
REMARKS	VARCHAR(254)	Yes	User-provided comment.

SYSCAT.TRIGDEP

Contains a row for every dependency of a trigger on some other object.

Table 77. SYSCAT.TRIGDEP Catalog View

Column Name	Data Type	Nullable	Description
TRIGSCHEMA	CHAR(8)		Qualified name of the trigger.
TRIGNAME	VARCHAR(18)		
BTYPE	CHAR(1)		Type of object that is depended on. A=Alias F=Function instance T=Table V=View
BSCHEMA	CHAR(8)		Qualified name of object depended on by a trigger.
BNAME	VARCHAR(18)		
TABAUTH	SMALLINT	Yes	If BTYPE=T or V, encodes the privileges on the table or view that are required by this trigger; otherwise null.

SYSCAT.TRIGGERS

Contains one row for each trigger.

Table 78. SYSCAT.TRIGGERS Catalog View

Column Name	Data Type	Nullable	Description
TRIGSCHEMA	CHAR(8)		Qualified name of the trigger.
TRIGNAME	VARCHAR(18)		
DEFINER	CHAR(8)		Authorization ID under which the trigger was defined.
TABSCHEMA	CHAR(8)		Qualified name of the table to which this trigger applies.
TABNAME	VARCHAR(18)		
TRIGTIME	CHAR(1)		Time when triggered actions are applied to the base table, relative to the event that fired the trigger: B=Trigger applied before event A=Trigger applied after event
TRIGEVENT	CHAR(1)		Event that fires the trigger. I=Insert D=Delete U=Update
GRANULARITY	CHAR(1)		Trigger is executed once per: S=Statement R=Row
VALID	CHAR(1)		Y=Trigger is valid X=Trigger is inoperative; must be re-created.
TEXT	CLOB(32K)		The full text of the CREATE TRIGGER statement, exactly as typed.
CREATE_TIME	TIMESTAMP		Time at which the trigger was defined. Used in resolving functions and types.
FUNC_PATH	VARCHAR(254)		Function path at the time the trigger was defined. Used in resolving functions and types.
REMARKS	VARCHAR(254)	Yes	User-supplied comment, or null.

SYSCAT.VIEWDEP

Contains a row for every dependency of a view on some other object. Also encodes how privileges on this view depend on privileges on underlying tables and views.

Table 79. SYSCAT.VIEWDEP Catalog View

Column Name	Data Type	Nullable	Description
VIEWSCHEMA	CHAR(8)		Name of the view.
VIEWNAME	VARCHAR(18)		
DEFINER	CHAR(8)	Yes	Authorization ID of the creator of the view.
BTYPE	CHAR(1)		Type of object that the specified view has a dependency on. T=Table V=View F=Function instance A=Alias
BSCHEMA	CHAR(8)		Qualified name of object depended on by the view.
BNAME	VARCHAR(18)		
TABAUTH	SMALLINT	Yes	Encodes the privileges on the underlying table or view that this view depends on. Otherwise null.

SYSCAT.VIEWS

Contains one or more rows for each view that is created.

Table 80. SYSCAT.VIEWS Catalog View

Column Name	Data Type	Nullable	Description
VIEWSCHEMA	CHAR(8)		Name of the view.
VIEWNAME	VARCHAR(18)		
DEFINER	CHAR(8)		Authorization ID of the creator of the view.
SEQNO	SMALLINT		Sequence number of this row; the first portion of the view is on row one, and successive rows have increasing values of SEQNO.
VIEWCHECK	CHAR(1)		States the type of view checking: N=No check option L=Local check option C=Cascaded check option
READONLY	CHAR(1)		Y=View is read-only because of its definition. N=View is not read-only.
VALID	CHAR(1)		Y=View definition is valid. X=View definition is inoperative; must be re-created.
FUNC_PATH	VARCHAR(254)		The function path of the view creator at the time the view was defined. When the view is used in data manipulation statements, this path must be used to resolve function calls in the view. SYSIBM for views created before Version 2.
TEXT	VARCHAR(3600)		Text or portion of the text of the CREATE VIEW statement.

SYSSTAT.COLDIST

Each row describes the Nth-most-frequent value or Nth quantile value of some column.

Table 81. SYSSTAT.COLDIST Catalog View

Column Name	Data Type	Nullable	Description	Updatable
TABSCHEMA	CHAR(8)		Qualified name of the table to which this entry applies.	
TABNAME	VARCHAR(18)			
COLNAME	VARCHAR(18)		Name of the column to which this entry applies.	
TYPE	CHAR(1)		Type of statistic collected: F=Frequency (most frequent value) Q=Quantile value	
SEQNO	SMALLINT		If TYPE=F, then N in this column identifies the Nth most frequent value. If TYPE=Q, then N in this column identifies the Nth quantile value.	
COLVALUE	VARCHAR(33)	Yes	The data value, as a character literal or a null value. This column can be updated with a valid representation of the value appropriate to the column that the statistic is associated with. If null is the required frequency value, the column should be set to NULL.	Yes
VALCOUNT	INTEGER		If TYPE=F, then VALCOUNT is the number of occurrences of COLVALUE in the column. If TYPE=Q, then VALCOUNT is the number of rows whose value is less than or equal to COLVALUE. This column can be only updated with the following values: • >= 0 (zero)	Yes
DISTCOUNT	INTEGER		If TYPE=q, this column records the number of distinct values that are less than or equal to COLVALUE (null iv unavailable.) the number of rows whose value is less than or equal to COLVALUE.	Yes

SYSSTAT.COLUMNS

Contains one row for each column that is defined for the specified table.

Table 82. SYSSTAT.COLUMNS Catalog View

Column Name	Data Type	Nullable	Description	Updatable
TABSCHEMA	CHAR(8)		Qualified name of the table that contains the column.	
TABNAME	VARCHAR(18)			
COLNAME	VARCHAR(18)		Column name.	
COLCARD	INTEGER		Number of distinct values in the column; –1 if statistics are not gathered. For any column, COLCARD cannot have a value higher than the cardinality of the table containing that column. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
HIGH2KEY	VARCHAR(33)		Second highest value of the column. This field is empty if statistics are not gathered. This column can be updated with a valid representation of the value appropriate to the column that the statistic is associated with. LOWKEY2 should not be greater than HIGH2KEY.	Yes
LOW2KEY	VARCHAR(33)		Second lowest value of the column. Empty if statistics not gathered. This column can be updated with a valid representation of the value appropriate to the column that the statistic is associated with.	Yes
AVGCOLLEN	INTEGER		Average column length. -1 if a long field or LOB, or statistics have not been collected. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes

SYSSTAT.FUNCTIONS

Contains a row for each user-defined function (scalar or aggregate). Does not include built-in functions.

Table 83 (Page 1 of 2). SYSSTAT.FUNCTIONS Catalog View

Column Name	Data Type	Nullable	Description	Updatable
FUNCSCHEMA	CHAR(8)		Qualified function name.	
FUNCNAME	VARCHAR(18)			
SPECIFICNAME	VARCHAR(18)		Function specific (instance) name.	
IOS_PER_INVOC	DOUBLE		Estimated number of I/Os per invocation; -1 if not known (0 default). This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
INSTS_PER_INVOC	DOUBLE		Estimated number of instructions per invocation; -1 if not known (450 default). This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
IOS_PER_ARGBYTE	DOUBLE		Estimated number of I/O's per input argument byte; -1 if not known (0 default). This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
INSTS_PER_ARGBYTE	DOUBLE		Estimated number of instructions per input argument byte; -1 if not known (0 default). This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
PERCENT_ARGBYTES	SMALLINT		Estimated average percent of input argument bytes that the function will actually read; -1 if not known (100 default). This column can only be updated with the following values: • -1 or between 100 and 0 (zero)	Yes
INITIAL_IOS	DOUBLE		Estimated number of I/O's performed the first/last time the function is invoked; -1 if not known (0 default). This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes

SYSSTAT.FUNCTIONS

Table 83 (Page 2 of 2). SYSSTAT.FUNCTIONS Catalog View

Column Name	Data Type	Nullable	Description	Updatable
INITIAL_INSTS	DOUBLE		Estimated number of instructions executed the first/last time the function is invoked; -1 if not known (0 default). This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
CARDINALITY	INTEGER		The predicted cardinality of a table function. –1 if not known, or if function is not a table function.	Yes

SYSSTAT.INDEXES

Contains one row for each index that is defined for a table.

Table 84 (Page 1 of 3). SYSSTAT.INDEXES Catalog View

Column Name	Data Type	Nullable	Description	Updatable
INDSCHEMA	CHAR(8)		Qualified name of the index.	
INDNAME	VARCHAR(18)			
NLEAF	INTEGER		Number of leaf pages; –1 if statistics are not gathered. This column can only be updated with the following values: • -1 or > 0 (zero)	Yes
NLEVELS	SMALLINT		Number of index levels; –1 if statistics are not gathered. This column can only be updated with the following values: • -1 or > 0 (zero)	Yes
FIRSTKEYCARD	INTEGER		Number of distinct first key values; –1 if statistics are not gathered. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
FIRST2KEYCARD	INTEGER		Number of distinct keys using the first two columns of the index (–1 if no statistics or inapplicable) This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
FIRST3KEYCARD	INTEGER		Number of distinct keys using the first three columns of the index (–1 if no statistics or inapplicable) This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
FIRST4KEYCARD	INTEGER		Number of distinct keys using the first four columns of the index (–1 if no statistics or inapplicable) This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes

SYSSTAT.INDEXES

Table 84 (Page 2 of 3). SYSSTAT.INDEXES Catalog View

Column Name	Data Type	Nullable	Description	Updatable
FULLKEYCARD	INTEGER		Number of distinct full key values; –1 if statistics are not gathered. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
CLUSTERRATIO	SMALLINT		This is used by the optimizer. It indicates the degree of data clustering with the index; –1 if statistics are not gathered or if detailed index statistics have been gathered. This column can only be updated with the following values: • –1 or between 0 and 100	Yes
CLUSTERFACTOR	DOUBLE		This is used by the optimizer. It is a finer measurement of degree of clustering, or -1 if detailed index statistics have not been gathered. This column can only be updated with the following values: • -1 or between 0 and 1	Yes
SEQUENTIAL_PAGES	INTEGER		Number of leaf pages located on disk in index key order with few or no large gaps between them. (–1 if no statistics are available.) This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
DENSITY	INTEGER		Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percent (integer between 0 and 100, –1 if no statistics are available.) This column can only be updated with the following values: • –1 or between 0 and 100	Yes

Table 84 (Page 3 of 3). SYSSTAT.INDEXES Catalog View

Column Name	Data Type	Nullable	Description	Updatable
PAGE_FETCH_PAIRS	VARCHAR(254)		A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer, and the number of page fetches required to scan the index using that hypothetical buffer. (Zero-length string if no data available.) This column can be updated with the following input values: • The pair delimiter and pair separator characters are the only non-numeric characters accepted • Blanks are the only characters recognized as a pair delimiter and pair separator • Each number entry must have an accompanying partner number entry with the two being separated by the pair separator character • Each pair must be separated from any other pairs by the pair delimiter character • Each expected number entry must between 0-9 (only positive values)	Yes

SYSSTAT.TABLES

Contains one row for each *base* table. Views or aliases are, therefore, not included.

Table 85. SYSSTAT.TABLES Catalog View

Column Name	Data Type	Nullable	Description	Updatable
TABSCHEMA	CHAR(8)		Qualified name of the table.	
TABNAME	VARCHAR(18)			
CARD	INTEGER		Total number of rows in the table; –1 if statistics are not gathered. An update to CARD for a table should not attempt to assign it a value less than the COLCARD value of any of the columns in that table. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
NPAGES	INTEGER		Total number of pages on which the rows of the table exist; –1 if statistics are not gathered. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
FPAGES	INTEGER		Total number of pages in the file; –1 if statistics are not gathered. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes
OVERFLOW	INTEGER		Total number of overflow records in the table; –1 if statistics are not gathered. This column can only be updated with the following values: • -1 or >= 0 (zero)	Yes

Appendix J. User Exit for Database Recovery

User exits allow you to develop your own user exit program to interact with storage devices that are not directly supported by the operating system.

The following topics describe the purpose of and considerations for a user exit program, and discuss the sample exit programs and error handling:

- Overview for OS/2
- Overview for UNIX-Based Operating Systems
- Invoking a User Exit Program
- Sample User Exit Programs
- Calling Format
- Archive and Retrieve Considerations
- Backup and Restore Considerations (DB2 for OS/2 only)
- Error Handling

As noted in the sections, some of the information may only be applicable to certain operating platforms. For example, backup and restore user exits are **not** applicable to UNIX-based platforms.

Overview for OS/2

The database manager can optionally call a user exit program to backup and restore a database, to archive and retrieve log files, or both. Calling a user exit program for one pair of tasks (backup and restore or archive and retrieve) does not require that a user exit program be used for the other pair of tasks. For example, if you archive and retrieve logs with a user exit program, you are not required to back up and restore databases with a user exit program.

The database manager can call a user exit program with one of the following actions:

Backup

The BACKUP DATABASE utility calls a user exit program when you specify `0:` as the target drive parameter from the command line processor, or `U` as the media type on the API call. Refer to “Backing Up a Database” on page 201 for additional information about backing up a database.

Restore

The RESTORE DATABASE utility calls a user exit program to retrieve database files that were previously stored by BACKUP DATABASE calling a user exit program. The RESTORE DATABASE utility calls a user exit program by specifying `0:` as the source drive parameter from the command line processor, or `U` as the media type on the API call. Refer to “Restoring a Database” on page 207 for additional information about restoring a database.

Archive and Retrieve

The database manager archive and retrieve functions call a user exit program to store and retrieve log files and to manage the location of

archived log files if the database configuration parameter, *userexit*, is on. Using a user exit program to archive and retrieve files enables a database for roll-forward recovery (refer to “Rolling Forward Changes in a Database” on page 216).

Note: The *userexit* configuration parameter applies to the archiving and retrieving of log files only.

Overview for UNIX-Based Operating Systems

The database manager can call a user exit program to store and retrieve log files and to manage the location of archived log files if the database configuration parameter, *userexit*, is on. Using a user exit program to archive and retrieve files enables a database for roll-forward recovery (refer to “Rolling Forward Changes in a Database” on page 216).

Invoking a User Exit Program

When the user exit program is invoked, the database manager passes control to the executable file, `[db2uext2]`.

Note: Backup and restore operations call `[db2usrxt.cmd]` first which in turn calls `[db2uext2]`.

The database manager passes parameters to this program, and on completion the program passes a return code back to the database manager. Because the database manager can only handle a limited set of return conditions, the user exit program should handle error conditions.

Only one user exit program can be invoked within a database manager instance. Therefore, each program must have sections for all of the actions it may need to perform, including: archive, retrieve, backup (OS/2 only) and restore (OS/2 only). One of the parameters passed to the user exit program indicates which of these actions is requested.

Sample User Exit Programs

A number of sample programs are provided to demonstrate the usage of the user exit function for a different device or software interface. The program listings identify the version of the device support software used.

You may modify or otherwise use these programs in any way you wish. Comments within these sample programs provide technical information for writing your own user exit programs.

The following topics provide information about the sample programs related to your operating system:

- Sample User Exit Programs for OS/2
- Sample User Exit Programs for UNIX-Based Operating Systems.

Sample User Exit Programs for OS/2

The user exit sample programs for DB2 for OS/2 are found in the instance subdirectory of the \sqllib\samples\rexx directory. The last user exit sample program (`dbuexit.CAD`) is an exception: it is found in the instance subdirectory of the \sqllib\samples\c directory. The sample you choose to implement should be renamed with the executable file name of `db2uexit` with an extension of either `.cmd` or `.exe`. This renamed file should be placed in the \sqllib\bin directory for use as a user exit program.

While the samples provided are mostly REXX command files, your user exit program can be written in a different programming language. The executable file name must be `db2uexit` with an extension of either `.cmd` or `.exe`.

There are five OS/2 sample programs provided:

- **db2uexit.ex1**

  This program uses the Sytos Premium** Version 2.2 program, available from the Seagate** Corporation, to store and retrieve data on an IBM external tape device.

  Note: We only support Version 2.2 of the Sytos Premium** product in this release.

  Review the sample program listing to determine requirements such as predefining procedures.
- **db2uexit.ex2**

  This program uses the Filesafe** program, available from the Mountain** Corporation, to store and retrieve data on a Mountain tape device.

  A unique volume label is assigned to each backup copy of a database so that multiple backups of the same database or different databases can be stored on the same tape. When a database is being restored, this program selects the most recent backup copy. This feature can be bypassed by modifying the backup log file.
- **db2uexit.ex3**

  This program uses the MaynStream** program, available from the Maynard** Corporation, to store and retrieve data on a Maynard tape device.

  MaynStream does not support redirecting the restored database to a drive other than the one on which the database was backed up.
- **db2uexit.ex4**

  This program uses the OS/2 XCOPY command. The storage device can be any device supported by OS/2, such as a fixed disk, diskette, or optical cartridge. These devices can be LAN redirected drives if the workstation is set up to support redirected drives.

  XCOPY cannot be used for backing up and restoring databases.
- **db2uexit.CAD**

  This C program is equivalent to the ADSTAR Distributed Storage Manager (ADSM) sample program to archive and retrieve database logfiles as presented in the sample programs for UNIX-based operating systems.

Sample User Exit Programs for UNIX-Based Operating Systems

The *userexit* configuration parameter causes the database manager to call a user exit program for archiving and retrieving logs. There are three IBM-supplied sample user exit programs on UNIX platforms: one for disk, one for tape, and one for ADSM. It is not mandatory that you use these programs. You may choose to create your own user exit programs. The sample programs may provide you with a model or suggestions that you can use when creating your user exit programs. Useful information is found in the header information in each sample program.

While the samples provided are coded in the C language, your user exit program can be written in a different programming language. The user exit program must be an executable file whose name is `db2uext2`.

There are three UNIX-based operating system sample programs provided:

- **db2uext2.cadsm**

  This program uses the ADSTAR Distributed Storage Manager utility to archive and retrieve database log files.

- **db2uext2.ctape**

  This program archives and retrieves the database log files using tape media.

- **db2uext2.cdisk**

  This program uses the operating system copy command to archive and retrieve database log files using disk media.

Calling Format

The database manager will call the user exit program as required and will pass a set of parameters to it. These parameters have a data type of character string or character.

The calling format is dependent on your operating environment as is described in the following topics:

- Calling Format for OS/2
- Calling Format for UNIX-Based or Windows NT Operating Systems.

Calling Format for OS/2

The following is the database manager format for calling an OS/2 user exit program:

```
action drive db_alias log_path log_file indicator
```

action Contains the value `BACKUP`, `RESTORE`, `ARCHIVE`, or `RETRIEVE`.

drive For BACKUP, this parameter contains the drive where the database to be backed up resides.

For RESTORE, this parameter contains the drive where the database is to be restored.

For ARCHIVE and RETRIEVE, this parameter contains the drive where the database is located.

The format of this parameter is the drive letter followed by a colon (for example, `C:`).

db_alias Contains the database alias, or, if no alias exists for the database, the database name.

log_path For BACKUP, this parameter contains a fully qualified name of a response file, which contains a list of files to be backed up. Each file name in the list is a fully qualified name and may contain wild cards.

For RESTORE, this parameter contains the fully qualified name of a response file, which is the list of files to be restored. Each file name in the list is a fully qualified name and may contain wild cards. The drive letter and path are the source drive and path at the time the database file was backed up. For example, if `C:\SQLUTIL\dbname.MH1` is contained in the response file, it means that the `dbname.MH1` file was backed up from `C:\SQLUTIL`.

For ARCHIVE and RETRIEVE, this parameter contains the log path directory (for example, `C:\SQL00001\SQLOGDIR\`).

log_file For BACKUP, this parameter contains a media label generated by the BACKUP DATABASE utility. This label is composed of the database alias name and timestamp.

For RESTORE, this parameter contains the path name of the database subdirectory where the files are to be restored. The drive letter is not included, because it is indicated in the *drive* parameter. The format is `\SQLnnnnn\`.

For ARCHIVE and RETRIEVE, this parameter contains the log file name (for example, `S0000001.LOG`).

indicator An indicator used to support multiple calls during a backup or restore operation. The first call has a value of the character `'1'`, and subsequent calls have a value of the character `'2'`.

The user exit program is called multiple times during a backup or restore operation. The first call backs up or restores media header files (the `.MH`*n* files), and the second call backs up or restores the entire set of database files.

For ARCHIVE and RETRIEVE, this parameter is not used.

Calling Format for UNIX-Based or Windows NT Operating Systems

The following is the database manager format for calling a UNIX-based or Windows NT operating system user exit program to archive or retrieve data:

```
db2uext2 -OS<os> -RL<db2rel> -RQ<request>
-DB<dbname> -NN<nodenum> -LP<logpath>
-LN<logname> -AP<adsmpasswd>
```

os	Platform on which the instance is running: `AIX`, `NT`, `SUN`, `HP`, `SNI`, `SCO`, `95`, and `SGI`.
db2rel	DB2 release level. For example, `DB2_V5.1.0` or `DB2_V5.1.1`.
request	Request type. This can be `ARCHIVE` or `RETRIEVE`.
dbname	Database name.
nodenum	Local node number, such as `5`.
logpath	Fully qualified path to the log files. The path must contain the trailing path separator. For example, `/u/database/log/path/` or `d:\logpath\`.
logname	Name of log file to be archived or retrieved, such as `S0000123.LOG`.
adsmpasswd	ADSM password. It will be passed to the user exit if it is provided in the database configuration.

Note: Windows NT only supports user exits for archiving logs.

Archive and Retrieve Considerations

The following considerations apply to calling a user exit program for archiving and retrieving log files:

- The database configuration file parameter *userexit* specifies whether the database manager invokes a user exit program to archive files or to retrieve log files during roll-forward recovery of databases. A request to retrieve a log file is made when the roll-forward database recovery utility needs a log file that is not found in the log path directory.

  Note: Table space roll-forward recovery does not support the retrieval of log files using user exits.

- When archiving, a log file is passed to the user exit when it is full, even if the log file is still active and is needed for normal processing. This allows copies of the data to be moved away from volatile media as quickly as possible. The log file passed to the user exit is retained in the log path directory until it is no longer needed for normal processing. At this point, the disk space is reused.

- A user exit program does not guarantee roll-forward recovery to the point of failure, but only attempts to make the failure window smaller. As log files fill, they are queued for the user exit routine. Should the disk containing the log fail before a log file is filled, the data in that log file is lost. Also, since the files are queued for archiving, the disk can fail before all the files are copied. Any log files in the queue are lost.

- The configured size of each individual log file has a direct bearing on the user exit. If each log file is very large, a large amount of data can be lost if a disk fails. A log file configured with small log files causes the data to be passed to the user exit routine more often.

  However, if you are moving the data to a slower device such as tape, you might want to have larger log files to prevent the queue from building up. If the queue

becomes full, archive and retrieve requests will not be processed. Processing will resume when there is room on the queue. Any requests not processed will not be automatically re-queued.

- An archive request to the user exit program occurs only when *userexit* is configured and each time an active log file is filled. It is possible that an active log file is not full when the last disconnection from the database occurs and the user exit program is also called for a partially filled active log file.

  Note: To free unused log space, the log file is truncated before it is archived.

- A copy of the log should be made to another physical device so that the off-line log file can be used by roll-forward recovery if the device containing the log file has a media failure. This should not be the same device containing the database data files.
- In some cases, if a database is closed before a positive response has been received from a user exit program for an archive request, the database manager will send another request when the database is opened. Thus, a log file may be archived more than once. If you do not want this multiple archiving to occur, the user exit program must not allow the subsequent requests for archiving the same file.
- If a user exit program receives a request to archive a file that does not exist (because there were multiple requests to archive and the file was deleted after the first successful archiving), or to retrieve a file that does not exist (because it is located in another directory or the end of the logs has been reached), it should ignore this request and return a successful return code.
- A user exit may be interrupted if a remote client loses its connection to the DB2 server. That is, while handling the archiving of logs through a user exit, one of the other SNA-connected clients dies or powers off resulting in a signal (SIGUSR1) being sent to the server. The server passes the signal to the user exit causing an interrupt. The user exit program can be modified to check for an interrupt and then continue.
- The user exit program should allow for the existence of different log files with the same name after a point-in-time recovery; it should be written to preserve both log files and to associate those log files with the correct recovery path. (See “Considerations for Managing Log Files” on page 227.)
- If two or more databases are using a device at the same time, and one of the operations involves a roll-forward operation, a log file needed for roll-forward recovery may not exist on the medium currently in the drive. Two conditions can occur:
  - If the user exit program passes a zero (successful) return code back to the database manager and the requested log file has not been retrieved, the database manager assumes the roll-forward operation is complete to the end of the logs, and the roll-forward operation stops. However, roll-forward processing may not have gone to the end of the logs.
  - If a non-zero return code is returned, the database will be in a roll-forward pending state, and you must either resume or stop roll-forward processing.

To prevent either situation from occurring, you can ensure that no other databases on the node that calls the user exit program are open during the roll-forward operation, or write a user exit program to handle this situation.

Backup and Restore Considerations (DB2 for OS/2 only)

The following considerations apply if you are writing a user exit program which is called from the BACKUP DATABASE and RESTORE DATABASE utilities:

- A non-zero return code returned by a user exit program causes the utility to fail, and no retry is attempted.
- A wild card must be supported in the file name of a fully qualified file name. For example, `C:\SQL00001\*.*` and `C:\*.MH*` are both acceptable search criteria.
- The user exit program must handle the response file format of one fully qualified file name per line with each line terminated by a carriage return and line feed. There is no end-of-file character in the file.
- If multiple backups of the same database are placed on one media, the user exit program should be designed so that the correct version of the backup will be selected during the restore operation. (See the **db2uexit.ex2** sample, as described in "Sample User Exit Programs for OS/2" on page 755.)
- Two concurrently running backup processes that are sharing one backup device must be serialized.
- If a backup image is spanned over more than one media, the prompting for the media must be handled by the user exit program or an application it may call. To support this feature, BACKUP DATABASE and RESTORE DATABASE open an operating system foreground session to call the user exit program.
- The user exit program must not back up any subdirectory within the database directory.
- When restoring a database using a user exit program, RESTORE DATABASE requires complete control over that database. However, the workstation can have active connections to databases other than the one being restored.
- If a database is being backed up or restored with a user exit program and another operation is using the same tape device, the backup or restore operation could fail. The backup or restore operation will have to be restarted. To avoid this situation, you can ensure that no other databases on the workstation that call the user exit program for logging are in use while a backup or restore operation is in progress, or you can ensure that the user exit program retries the backup or restore operation at a later time if a device is not ready.
- During the restore operation, the drive letter and the path can be different from those specified during the backup operation. For example, if file `dbname.MH1` is backed up from `C:\SQLUTIL`, you can restore it into `d:\xxx`.

Error Handling

In order for the database manager to properly handle the return codes from the user exit program, the program must be coded to provide specific return codes to show specific results.

Table 86 on page 761 shows the return codes that can be returned by a user exit program. and how the database manager interprets that return code. If a return code is not listed in the table, it is treated as if its value were 32.

Table 86. User Exit Return Codes and Results

Return Code	Result (Note 1)	Explanation
0	—	Successful.
4	Note 2	Temporary resource error encountered.
8	Note 2	Operator intervention is required.
12	Note 3	Hardware error.
16	Note 3	Error with the user exit program or a software function used by the program.
20	Note 3	Error with one or more of the parameters passed to the user exit program. Verify that the user exit program is correctly processing the parameters provided.
24	Note 3	The user exit program was not found. For OS/2 this error message also means that a file needed to complete a RESTORE DATABASE operation could not be found in the current backup media.
28	Note 3	Error caused by an I/O failure or the operating system.
32 (and all other values)	Note 3	The user exit program was terminated by the user.

Notes:

1. Applies to archive and retrieve actions only.
2. For archive and retrieve, a return code of 4 or 8 causes a retry in five minutes.
3. No further user exit program requests will be sent for this database while the database is open for processing. If all applications disconnect from the database and then the database is reopened, the request will be repeated.

   If the user exit program was called to archive log files, your disk can be filled with log files and performance may be degraded because of extra work to format these log files. Once the disk becomes full, database manager will not accept further application requests for database changes.

   If the user exit program was called to retrieve log files, roll-forward recovery is suspended but not stopped unless a stop was specified in the ROLLFORWARD DATABASE utility. If a stop was not specified, you can correct the problem and resume recovery.
4. For archive and retrieve actions, an alert message is issued for all return codes except 0, 4, and 24. The alert message contains the return code from the user exit program and a copy of the input parameters that were provided to the user exit program.

Because the user exit program is called by the underlying operating system command processor, there is a possibility that non-zero return codes are returned from the operating system. These error codes are not remapped. Consult the operating system message help information for a description of those error codes.

Error Handling for OS/2:

For the BACKUP DATABASE and RESTORE DATABASE utilities, any non-zero return code returned by a user exit program causes the utility to fail and no retry is attempted. The utilities report a general SQLCODE -2029. The message text for this SQLCODE displays the return code returned from the user exit program or from the operating system.

Appendix K. Explain Tables and Definitions

The Explain tables capture access plans when the Explain facility is activated. The following Explain tables and definitions are described in this section:

The Explain tables must be created before Explain can be invoked. To create them, use the sample command line processor input script provided in the EXPLAIN.DDL file located in the `'misc'` subdirectory of the `'sqllib'` directory. Connect to the database where the Explain tables are required. Then issue the command: `db2 -tf EXPLAIN.DDL` and the tables will be created. See "Table Definitions for Explain Tables" on page 776 for more information.

The population of the Explain tables by the Explain facility will neither activate any triggers nor activate any referential or check constraints. For example, if an insert trigger were defined on the EXPLAIN_INSTANCE table and an eligible statement were explained, the trigger would not be activated.

See Chapter 13, "SQL Explain Facility" on page 377 for more details on the Explain facility.

Legend for the Explain Tables:

Heading	Explanation
Column name	Name of the column
Data Type	Data type of the column
Nullable?	Yes: Nulls are permitted
	No: Nulls are not permitted
Key?	PK: Column is part of a primary key
	FK: Column is part of a foreign key
Description	Description of the column

EXPLAIN_ARGUMENT Table

The EXPLAIN_ARGUMENT table represents the unique characteristics for each individual operator, if there are any.

Table 87. EXPLAIN_ARGUMENT Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	CHAR(8)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	CHAR(8)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when static SQL was explained.
SOURCE_SCHEMA	CHAR(8)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	SMALLINT	No	FK	Statement number within package to which this explain information is related.
SECTNO	SMALLINT	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	SMALLINT	No	No	Unique ID for this operator within this query.
ARGUMENT_TYPE	CHAR(8)	No	No	The type of argument for this operator.
ARGUMENT_VALUE	VARCHAR(30)	No	No	The value of the argument for this operator.

Table 88 (Page 1 of 3). ARGUMENT_TYPE and ARGUMENT_VALUE Column Values

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
AGGMODE	COMPLETE PARTIAL INTERMEDIATE FINAL	Partial aggregation indicators.
CSETEMP	TRUE FALSE	Temporary Table over Common Subexpression Flag.
DIRECT	TRUE	Direct fetch indicator.
DUPLWARN	TRUE FALSE	Duplicates Warning flag.
EARLYOUT	TRUE FALSE	Early out indicator.
FETCHMAX	IGNORE INTEGER	Override value for MAXPAGES argument on FETCH operator.
GROUPBYC	TRUE FALSE	Whether Group By columns were provided.
GROUPBYN	Integer	Number of comparison columns.
GROUPBYR	Each row of this type will contain: • Ordinal value of column in group by clause (followed by a colon and a space) • Name of Column	Group By requirement.

Table 88 (Page 2 of 3). ARGUMENT_TYPE and ARGUMENT_VALUE Column Values

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
INNERCOL	Each row of this type will contain: • Ordinal value of column in order (followed by a colon and a space) • Name of Column • Order Value **(A)** Ascending **(D)** Descending	Inner order columns.
ISCANMAX	IGNORE INTEGER	Override value for MAXPAGES argument on ISCAN operator.
JN_INPUT	INNER OUTER	Indicates if operator is the operator feeding the inner or outer of a join.
LISTENER	TRUE FALSE	Listener Table Queue indicator.
MAXPAGES	ALL NONE INTEGER	Maximum pages expected for Prefetch.
NUMROWS	INTEGER	Number of rows expected to be sorted.
ONEFETCH	TRUE FALSE	One Fetch indicator.
OUTERCOL	Each row of this type will contain: • Ordinal value of column in order (followed by a colon and a space) • Name of Column • Order Value **(A)** Ascending **(D)** Descending	Outer order columns.
OUTERJN	LEFT RIGHT	Outer join indicator.
PARTCOLS	Name of Column	Partitioning columns for operator.
PREFETCH	LIST NONE SEQUENTIAL	Type of Prefetch Eligible.
ROWLOCK	EXCLUSIVE NONE REUSE SHARE SHORT (INSTANT) SHARE UPDATE	Row Lock Intent.
ROWWIDTH	INTEGER	Width of row to be sorted. ***
SCANDIR	FORWARD REVERSE	Scan Direction.
SHARED	TRUE	Intra-partition parallelism, shared TEMP indicator.
SCANGRAN	INTEGER	Intra-partition parallelism, granularity of the intra-partition parallel scan, expressed in SCANUNITs.
SCANTYPE	LOCAL PARALLEL	intra-partition parallelism, Index or Table scan.

Table 88 (Page 3 of 3). ARGUMENT_TYPE and ARGUMENT_VALUE Column Values

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
SCANUNIT	ROW PAGE	Intra-partition parallelism, scan granularity unit.
SLOWMAT	TRUE FALSE	Slow Materialization flag.
SORTKEY	Each row of this type will contain: • Ordinal value of column in key (followed by a colon and a space) • Name of Column • Order Value **(A)** Ascending **(D)** Descending	Sort key columns.
TABLOCK	EXCLUSIVE INTENT EXCLUSIVE INTENT NONE INTENT SHARE REUSE SHARE SHARE INTENT EXCLUSIVE SUPER EXCLUSIVE UPDATE	Table Lock Intent.
TQDEGREE	INTEGER	intra-partition parallelism, number of subagents accessing Table Queue.
TQMERGE	TRUE FALSE	Merging (sorted) Table Queue indicator.
TQREAD	READ AHEAD STEPPING SUBQUERY STEPPING	Table Queue reading property.
TQSEND	BROADCAST DIRECTED SCATTER SUBQUERY DIRECTED	Table Queue send property.
TQTYPE	LOCAL	intra-partition parallelism, Table Queue.
UNIQUE	TRUE FALSE	Uniqueness indicator.
UNIQKEY	Each row of this type will contain: • Ordinal value of column in key (followed by a colon and a space) • Name of Column	Unique key columns.

EXPLAIN_INSTANCE Table

The EXPLAIN_INSTANCE table is the main control table for all Explain information. Each row of data in the Explain tables is explicitly linked to one unique row in this table. The EXPLAIN_INSTANCE table gives basic information about the source of the SQL statements being explained as well as information about the environment in which the explanation took place.

For the definition of this table, see “EXPLAIN_INSTANCE Table Definition” on page 777.

Table 89 (Page 1 of 2). EXPLAIN_INSTANCE Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	CHAR(8)	No	PK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	PK	Time of initiation for Explain request.
SOURCE_NAME	CHAR(8)	No	PK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	CHAR(8)	No	PK	Schema, or qualifier, of source of Explain request.
EXPLAIN_OPTION	CHAR(1)	No	No	Indicates what Explain Information was requested for this request. Possible values are: **P** PLAN SELECTION
SNAPSHOT_TAKEN	CHAR(1)	No	No	Indicates whether an Explain Snapshot was taken for this request. Possible values are: **Y** Yes, an Explain Snapshot(s) was taken and stored in the EXPLAIN_STATEMENT table. Regular Explain information was also captured. **N** No Explain Snapshot was taken. Regular Explain information was captured. **O** Only an Explain Snapshot was taken. Regular Explain information was not captured.
DB2_VERSION	CHAR(7)	No	No	Product release number for DB2 Universal Database which processed this explain request. Format is vv.rr.m, where: **vv** Version Number **rr** Release Number **m** Maintenance Release Number
SQL_TYPE	CHAR(1)	No	No	Indicates whether the Explain Instance was for static or dynamic SQL. Possible values are: **S** Static SQL **D** Dynamic SQL
QUERYOPT	INTEGER	No	No	Indicates the query optimization class used by the SQL Compiler at the time of the Explain invocation. The value indicates what level of query optimization was performed by the SQL Compiler for the SQL statements being explained.
BLOCK	CHAR(1)	No	No	Indicates what type of cursor blocking was used when compiling the SQL statements. For more information, see the BLOCK column in SYSCAT.PACKAGES. Possible values are: **N** No Blocking **U** Block Unambiguous Cursors **B** Block All Cursors

Table 89 (Page 2 of 2). EXPLAIN_INSTANCE Table

Column Name	Data Type	Nullable?	Key?	Description
ISOLATION	CHAR(2)	No	No	Indicates what type of isolation was used when compiling the SQL statements. For more information, see the ISOLATION column in SYSCAT.PACKAGES. Possible values are: **RR** Repeatable Read **RS** Read Stability **CS** Cursor Stability **UR** Uncommitted Read
BUFFPAGE	INTEGER	No	No	Contains the value of the BUFFPAGE database configuration setting at the time of the Explain invocation.
AVG_APPLS	INTEGER	No	No	Contains the value of the AVG_APPLS configuration parameter at the time of the Explain invocation.
SORTHEAP	INTEGER	No	No	Contains the value of the SORTHEAP database configuration setting at the time of the Explain invocation.
LOCKLIST	INTEGER	No	No	Contains the value of the LOCKLIST database configuration setting at the time of the Explain invocation.
MAXLOCKS	SMALLINT	No	No	Contains the value of the MAXLOCKS database configuration setting at the time of the Explain invocation.
LOCKS_AVAIL	INTEGER	No	No	Contains the number of locks assumed to be available by the optimizer for each user. (Derived from LOCKLIST and MAXLOCKS.)
CPU_SPEED	DOUBLE	No	No	Contains the value of the CPUSPEED database manager configuration setting at the time of the Explain invocation.
REMARKS	VARCHAR(254)	Yes	No	User-provided comment.
DBHEAP	INTEGER	No	No	Contains the value of the DBHEAP database configuration setting at the time of Explain invocation.
COMM_SPEED	DOUBLE	No	No	Contains the value of the COMM_BANDWIDTH database configuration setting at the time of Explain invocation.
PARALLELISM	CHAR(2)	No	No	Possible values are: **N** No parallelism **P** Intra-partition parallelism **IP** Inter-partition parallelism **BP** Intra-partition parallelism and inter-partition parallelism

EXPLAIN_OBJECT Table

The EXPLAIN_OBJECT table identifies those data objects required by the access plan generated to satisfy the SQL statement.

Table 90 (Page 1 of 3). EXPLAIN_OBJECT Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	CHAR(8)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.

Table 90 (Page 2 of 3). EXPLAIN_OBJECT Table

Column Name	Data Type	Nullable?	Key?	Description
SOURCE_NAME	CHAR(8)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	CHAR(8)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	SMALLINT	No	FK	Statement number within package to which this explain information is related.
SECTNO	SMALLINT	No	FK	Section number within package to which this explain information is related.
OBJECT_SCHEMA	CHAR(8)	No	No	Schema to which this object belongs.
OBJECT_NAME	VARCHAR(18)	No	No	Name of the object.
OBJECT_TYPE	CHAR(2)	No	No	Descriptive label for the type of object.
CREATE_TIME	TIMESTAMP	Yes	No	Time of Object's creation; null if a table function.
STATISTICS_TIME	TIMESTAMP	Yes	No	Last time of update to statistics for this object; null if statistics do not exist for this object.
COLUMN_COUNT	SMALLINT	No	No	Number of columns in this object.
ROW_COUNT	INTEGER	No	No	Estimated number of rows in this object.
WIDTH	INTEGER	No	No	The average width of the object in bytes. Set to -1 for an index.
PAGES	INTEGER	No	No	Estimated number of pages that the object occupies in the buffer pool. Set to -1 for a table function.
DISTINCT	CHAR(1)	No	No	Indicates if the rows in the object are distinct (i.e. no duplicates) Possible values are: **Y** Yes **N** No
TABLESPACE_NAME	VARCHAR(18)	Yes	No	Name of the table space in which this object is stored; set to null if no table space is involved.
OVERHEAD	DOUBLE	No	No	Total estimated overhead, in milliseconds, for a single random I/O to the specified table space. Includes controller overhead, disk seek, and latency times. Set to -1 if no table space is involved.
TRANSFER_RATE	DOUBLE	No	No	Estimated time to read a data page, in milliseconds, from the specified table space. Set to -1 if no table space is involved.
PREFETCHSIZE	INTEGER	No	No	Number of data pages to be read when prefetch is performed. Set to -1 for a table function.
EXTENTSIZE	INTEGER	No	No	Size of extent, in data pages. This many pages are written to one container in the table space before switching to the next container. Set to -1 for a table function.
CLUSTER	DOUBLE	No	No	Degree of data clustering with the index. If >= 1, this is the CLUSTERRATIO. If >= 0 and < 1, this is the CLUSTERFACTOR. Set to -1 for a table, table function, or if this statistic is not available.
NLEAF	INTEGER	No	No	Number of leaf pages this index object's values occupy. Set to -1 for a table, table function, or if this statistic is not available.

Table 90 (Page 3 of 3). EXPLAIN_OBJECT Table

Column Name	Data Type	Nullable?	Key?	Description
NLEVELS	INTEGER	No	No	Number of index levels in this index object's tree. Set to -1 for a table, table function, or if this statistic is not available.
FULLKEYCARD	INTEGER	No	No	Number of distinct full key values contained in this index object. Set to -1 for a table, table function, or if this statistic is not available.
OVERFLOW	INTEGER	No	No	Total number of overflow records in the table. Set to -1 for an index, table function, or if this statistic is not available.

Table 91. Possible OBJECT_TYPE Values

Value	Description
IX	Index
TA	Table
TF	Table Function

EXPLAIN_OPERATOR Table

The EXPLAIN_OPERATOR table contains all the operators needed to satisfy the SQL statement by the SQL compiler.

Table 92 (Page 1 of 2). EXPLAIN_OPERATOR Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	CHAR(8)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	CHAR(8)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	CHAR(8)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	SMALLINT	No	FK	Statement number within package to which this explain information is related.
SECTNO	SMALLINT	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	SMALLINT	No	No	Unique ID for this operator within this query.
OPERATOR_TYPE	CHAR(6)	No	No	Descriptive label for the type of operator.
TOTAL_COST	DOUBLE	No	No	Estimated cumulative total cost (in instructions) of executing the chosen access plan up to and including this operator.
IO_COST	DOUBLE	No	No	Estimated cumulative I/O cost (in data page I/Os) of executing the chosen access plan up to and including this operator.
CPU_COST	DOUBLE	No	No	Estimated cumulative CPU cost (in instructions) of executing the chosen access plan up to and including this operator.

Table 92 (Page 2 of 2). EXPLAIN_OPERATOR Table

Column Name	Data Type	Nullable?	Key?	Description
FIRST_ROW_COST	DOUBLE	No	No	Estimated cumulative cost (in timerons) of fetching the first row for the access plan up to and including this operator. This value includes any initial overhead required.
RE_TOTAL_COST	DOUBLE	No	No	Estimated cumulative cost (in timerons) of fetching the next row for the chosen access plan up to and including this operator.
RE_IO_COST	DOUBLE	No	No	Estimated cumulative I/O cost (in data page I/Os) of fetching the next row for the chosen access plan up to and including this operator.
RE_CPU_COST	DOUBLE	No	No	Estimated cumulative CPU cost (in timerons) of fetching the next row for the chosen access plan up to and including this operator.
COMM_COST	DOUBLE	No	No	Estimated cumulative communication cost (in TCP/IP frames) of executing the chosen access plan up to and including this operator.
FIRST_COMM_COST	DOUBLE	No	No	Estimated cumulative communications cost (in TCP/IP frames) of fetching the first row for the chosen access plan up to and including this operator. This value includes any initial overhead required.
NODES_USED	CLOB(64K)	Yes	No	Cumulative list of nodes involved in executing the chosen access plan up to and including this operator.

Table 93. OPERATOR_TYPE Values

Value	Description
DELETE	Delete
FETCH	Fetch
FILTER	Filter rows
GENROW	Generate Row
GRPBY	Group By
INSERT	Insert
IXAND	Dynamic Bitmap Index ANDing
IXSCAN	Index Scan
MSJOIN	Merge Scan Join
NLJOIN	Nested loop Join
RETURN	Result
RIDSCN	Row Identifier (RID) Scan
SORT	Sort
TBSCAN	Table Scan
TEMP	Temporary Table Construction
TQ	Table Queue
UNION	Union
UNIQUE	Duplicate Elimination
UPDATE	Update

EXPLAIN_PREDICATE Table

The EXPLAIN_PREDICATE table identifies which predicates are applied by a specific operator.

Table 94. EXPLAIN_PREDICATE Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	CHAR(8)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	CHAR(8)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	CHAR(8)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	SMALLINT	No	FK	Statement number within package to which this explain information is related.
SECTNO	SMALLINT	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	SMALLINT	No	No	Unique ID for this operator within this query.
PREDICATE_ID	SMALLINT	No	No	Unique ID for this predicate for the specified operator.
HOW_APPLIED	CHAR(5)	No	No	How predicate is being used by the specified operator.
WHEN_EVALUATED	CHAR(3)	No	No	Indicates when the subquery used in this predicate is evaluated. Possible values are: **blank** This predicate does not contain a subquery. **EAA** The subquery used in this predicate is evaluated at application (EAA). That is, it is re-evaluated for every row processed by the specified operator, as the predicate is being applied. **EAO** The subquery used in this predicate is evaluated at open (EAO). That is, it is re-evaluated only once for the specified operator, and its results are re-used in the application of the predicate for each row. **MUL** There is more than one type of subquery in this predicate.
RELOP_TYPE	CHAR(2)	No	No	The type of relational operator used in this predicate.
SUBQUERY	CHAR(1)	No	No	Whether or not a data stream from a subquery is required for this predicate. There may be multiple subquery streams required. Possible values are: **N** No subquery stream is required **Y** One or more subquery streams is required
FILTER_FACTOR	DOUBLE	No	No	The estimated fraction of rows that will be qualified by this predicate.
PREDICATE_TEXT	CLOB(64K)	Yes	No	The text of the predicate as recreated from the internal representation of the SQL statement. Null if not available.

Table 95. Possible HOW_APPLIED Values

Value	Description
JOIN	Used to join tables
RESID	Evaluated as a residual predicate
SARG	Evaluated as a sargable predicate for index or data page
START	Used as a start condition
STOP	Used as a stop condition

Table 96. Possible RELOP_TYPE Values

Value	Description
blanks	Not Applicable
EQ	Equals
GE	Greater Than or Equal
GT	Greater Than
IN	In list
LE	Less Than or Equal
LK	Like
LT	Less Than
NE	Not Equal
NL	Is Null
NN	Is Not Null

EXPLAIN_STATEMENT Table

The EXPLAIN_STATEMENT table contains the text of the SQL statement as it exists for the different levels of Explain information. The original SQL statement as entered by the user is stored in this table along with the version used (by the optimizer) to choose an access plan to satisfy the SQL statement. The latter version may bear little resemblance to the original as it may have been rewritten and/or enhanced with additional predicates as determined by the SQL Compiler.

For the definition of this table, see “EXPLAIN_STATEMENT Table Definition” on page 781.

Table 97 (Page 1 of 3). EXPLAIN_STATEMENT Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	CHAR(8)	No	PK, FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	PK, FK	Time of initiation for Explain request.
SOURCE_NAME	CHAR(8)	No	PK, FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.

Table 97 (Page 2 of 3). EXPLAIN_STATEMENT Table

Column Name	Data Type	Nullable?	Key?	Description
SOURCE_SCHEMA	CHAR(8)	No	PK, FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	PK	Level of Explain information for which this row is relevant. Valid values are: **O** Original Text (as entered by user) **P** PLAN SELECTION
STMTNO	SMALLINT	No	PK	Statement number within package to which this explain information is related. Set to 1 for dynamic Explain SQL statements. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS catalog view.
SECTNO	SMALLINT	No	PK	Section number within package that contains this SQL statement. For dynamic Explain SQL statements, this is the section number used to hold the section for this statement at runtime. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS catalog view.
QUERYNO	INTEGER	No	No	Numeric identifier for explained SQL statement. For dynamic SQL statements (excluding the EXPLAIN SQL statement) issued through CLP or CLI, the default value is a sequentially incremented value. Otherwise, the default value is the value of STMTNO for static SQL statements and 1 for dynamic SQL statements.
QUERYTAG	CHAR(20)	No	No	Identifier tag for each explained SQL statement. For dynamic SQL statements issued through CLP (excluding the EXPLAIN SQL statement), the default value is `'CLP'`. For dynamic SQL statements issued through CLI (excluding the EXPLAIN SQL statement), the default value is `'CLI'`. Otherwise, the default value used is blanks.
STATEMENT_TYPE	CHAR(2)	No	No	Descriptive label for type of query being explained. Possible values are: **S** Select **D** Delete **DC** Delete where current of cursor **I** Insert **U** Update **UC** Update where current of cursor
UPDATABLE	CHAR(1)	No	No	Indicates if this statement is considered updatable. This is particularly relevant to SELECT statements which may be determined to be potentially updatable. Possible values are: **' '** Not applicable (blank) **N** No **Y** Yes
DELETABLE	CHAR(1)	No	No	Indicates if this statement is considered deletable. This is particularly relevant to SELECT statements which may be determined to be potentially deletable. Possible values are: **' '** Not applicable (blank) **N** No **Y** Yes

Table 97 (Page 3 of 3). EXPLAIN_STATEMENT Table

Column Name	Data Type	Nullable?	Key?	Description
TOTAL_COST	DOUBLE	No	No	Estimated total cost (in timerons) of executing the chosen access plan for this statement; set to 0 (zero) if EXPLAIN_LEVEL is *O* (original text) since no access plan has been chosen at this time.
STATEMENT_TEXT	CLOB(64K)	No	No	Text or portion of the text of the SQL statement being explained. The text shown for the Plan Selection level of Explain has been reconstructed from the internal representation and is SQL-like in nature; that is, the reconstructed statement is not guaranteed to follow correct SQL syntax.
SNAPSHOT	BLOB(10M)	Yes	No	Snapshot of internal representation for this SQL statement at the Explain_Level shown. This column is intended for use with DB2 Visual Explain. Column is set to null if EXPLAIN_LEVEL is *0* (original statement) since no access plan has been chosen at the time that this specific version of the statement is captured.
QUERY_DEGREE	INTEGER	No	No	Indicates the degree of intra-partition parallelism at the time of Explain invocation. For the original statement, this contains the directed degree of intra-partition parallelism. For the PLAN SELECTION, this contains the degree of intra-partition parallelism generated for the plan to use.

EXPLAIN_STREAM Table

The EXPLAIN_STREAM table represents the input and output data streams between individual operators and data objects. The data objects themselves are represented in the EXPLAIN_OBJECT table. The operators involved in a data stream are to be found in the EXPLAIN_OPERATOR table.

Table 98 (Page 1 of 2). EXPLAIN_STREAM Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	CHAR(8)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	CHAR(8)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	CHAR(8)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	SMALLINT	No	FK	Statement number within package to which this explain information is related.
SECTNO	SMALLINT	No	FK	Section number within package to which this explain information is related.
STREAM_ID	SMALLINT	No	No	Unique ID for this data stream within the specified operator.
SOURCE_TYPE	CHAR(1)	No	No	Indicates the source of this data stream: **O** Operator **D** Data Object

Table 98 (Page 2 of 2). EXPLAIN_STREAM Table

Column Name	Data Type	Nullable?	Key?	Description
SOURCE_ID	SMALLINT	No	No	Unique ID for the operator within this query that is the source of this data stream. Set to -1 if SOURCE_TYPE is 'D'.
TARGET_TYPE	CHAR(1)	No	No	Indicates the target of this data stream: **O** Operator **D** Data Object
TARGET_ID	SMALLINT	No	No	Unique ID for the operator within this query that is the target of this data stream. Set to -1 if TARGET_TYPE is 'D'.
OBJECT_SCHEMA	CHAR(8)	Yes	No	Schema to which the affected data object belongs. Set to null if both SOURCE_TYPE and TARGET_TYPE are 'O'.
OBJECT_NAME	VARCHAR(18)	Yes	No	Name of the object that is the subject of data stream. Set to null if both SOURCE_TYPE and TARGET_TYPE are 'O'.
STREAM_COUNT	DOUBLE	No	No	Estimated cardinality of data stream.
COLUMN_COUNT	SMALLINT	No	No	Number of columns in data stream.
PREDICATE_ID	SMALLINT	No	No	If this stream is part of a subquery for a predicate, the predicate ID will be reflected here, otherwise the column is set to -1.
COLUMN_NAMES	CLOB(64K)	Yes	No	This column contains the names and ordering information of the columns involved in this stream. These names will be in the format of: `NAME1(A)+NAME2(D)+NAME3+NAME4` Where *(A)* indicates a column in ascending order, *(D)* indicates a column in descending order, and no ordering information indicates that either the column is not ordered or ordering is not relevant.
PMID	SMALLINT	No	No	Partitioning map ID.
SINGLE_NODE	CHAR(5)	Yes	No	Indicates if this data stream is on a single or multiple partitions: **MULT** On multiple partitions **COOR** On coordinator node **HASH** Directed using hashing **RID** Directed using the row ID **FUNC** Directed using a function (PARTITION() or NODENUMBER()) **CORR** Directed using a correlation value
PARTITION_COLUMNS	CLOB(64K)	Yes	No	List of columns this data stream is partitioned on.

Table Definitions for Explain Tables

The Explain tables must be created before Explain can be invoked. The following definitions specify how to create the necessary Explain tables:

- "EXPLAIN_ARGUMENT Table Definition" on page 777
- "EXPLAIN_INSTANCE Table Definition" on page 777

Alternately, create them by using the sample command line processor input script provided in the EXPLAIN.DDL file located in the 'misc' subdirectory of the 'sqllib' directory. Connect to the database where the Explain tables are required. Then issue the command: `db2 -tf EXPLAIN.DDL` and the tables will be created.

EXPLAIN_ARGUMENT Table Definition

```
CREATE TABLE EXPLAIN_ARGUMENT ( EXPLAIN_REQUESTER CHAR(8)      NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                SOURCE_NAME       CHAR(8)      NOT NULL,
                                SOURCE_SCHEMA     CHAR(8)      NOT NULL,
                                EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                STMTNO            SMALLINT     NOT NULL,
                                SECTNO            SMALLINT     NOT NULL,
                                OPERATOR_ID       SMALLINT     NOT NULL,
                                ARGUMENT_TYPE     CHAR(8)      NOT NULL,
                                ARGUMENT_VALUE    VARCHAR(30)  NOT NULL,
                                   FOREIGN KEY (EXPLAIN_REQUESTER,
                                                EXPLAIN_TIME,
                                                SOURCE_NAME,
                                                SOURCE_SCHEMA,
                                                EXPLAIN_LEVEL,
                                                STMTNO,
                                                SECTNO)
                                   REFERENCES EXPLAIN_STATEMENT
                                   ON DELETE CASCADE )
```

EXPLAIN_INSTANCE Table Definition

Explain Tables

```
CREATE TABLE EXPLAIN_INSTANCE ( EXPLAIN_REQUESTER CHAR(8)   NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP NOT NULL,
                                SOURCE_NAME       CHAR(8)   NOT NULL,
                                SOURCE_SCHEMA     CHAR(8)   NOT NULL,
                                EXPLAIN_OPTION    CHAR(1)   NOT NULL,
                                SNAPSHOT_TAKEN    CHAR(1)   NOT NULL,
                                DB2_VERSION       CHAR(7)   NOT NULL,
                                SQL_TYPE          CHAR(1)   NOT NULL,
                                QUERYOPT          INTEGER   NOT NULL,
                                BLOCK             CHAR(1)   NOT NULL,
                                ISOLATION         CHAR(2)   NOT NULL,
                                BUFFPAGE          INTEGER   NOT NULL,
                                AVG_APPLS         INTEGER   NOT NULL,
                                SORTHEAP          INTEGER   NOT NULL,
                                LOCKLIST          INTEGER   NOT NULL,
                                MAXLOCKS          SMALLINT  NOT NULL,
                                LOCKS_AVAIL       INTEGER   NOT NULL,
                                CPU_SPEED         DOUBLE    NOT NULL,
                                REMARKS           VARCHAR(254),
                                DBHEAP            INTEGER   NOT NULL,
                                COMM_SPEED        DOUBLE    NOT NULL,
                                PARALLELISM       CHAR(2)   NOT NULL,
                                    PRIMARY KEY (EXPLAIN_REQUESTER,
                                                 EXPLAIN_TIME,
                                                 SOURCE_NAME,
                                                 SOURCE_SCHEMA))
```

EXPLAIN_OBJECT Table Definition

```
CREATE TABLE EXPLAIN_OBJECT ( EXPLAIN_REQUESTER CHAR(8)     NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP   NOT NULL,
                              SOURCE_NAME       CHAR(8)     NOT NULL,
                              SOURCE_SCHEMA     CHAR(8)     NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)     NOT NULL,
                              STMTNO            SMALLINT    NOT NULL,
                              SECTNO            SMALLINT    NOT NULL,
                              OBJECT_SCHEMA     CHAR(8)     NOT NULL,
                              OBJECT_NAME       VARCHAR(18) NOT NULL,
                              OBJECT_TYPE       CHAR(2)     NOT NULL,
                              CREATE_TIME       TIMESTAMP,
                              STATISTICS_TIME   TIMESTAMP,
                              COLUMN_COUNT      SMALLINT    NOT NULL,
                              ROW_COUNT         INTEGER     NOT NULL,
                              WIDTH             INTEGER     NOT NULL,
                              PAGES             INTEGER     NOT NULL,
                              DISTINCT          CHAR(1)     NOT NULL,
                              TABLESPACE_NAME   VARCHAR(18),
                              OVERHEAD          DOUBLE      NOT NULL,
                              TRANSFER_RATE     DOUBLE      NOT NULL,
                              PREFETCHSIZE      INTEGER     NOT NULL,
                              EXTENTSIZE        INTEGER     NOT NULL,
                              CLUSTER           DOUBLE      NOT NULL,
                              NLEAF             INTEGER     NOT NULL,
                              NLEVELS           INTEGER     NOT NULL,
                              FULLKEYCARD       INTEGER     NOT NULL,
                              OVERFLOW          INTEGER     NOT NULL,
                                      FOREIGN KEY (EXPLAIN_REQUESTER,
                                                   EXPLAIN_TIME,
                                                   SOURCE_NAME,
                                                   SOURCE_SCHEMA,
                                                   EXPLAIN_LEVEL,
                                                   STMTNO,
                                                   SECTNO)
                                   REFERENCES EXPLAIN_STATEMENT
                                   ON DELETE CASCADE )
```

EXPLAIN_OPERATOR Table Definition

```
CREATE TABLE EXPLAIN_OPERATOR ( EXPLAIN_REQUESTER CHAR(8)     NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP   NOT NULL,
                                SOURCE_NAME       CHAR(8)     NOT NULL,
                                SOURCE_SCHEMA     CHAR(8)     NOT NULL,
                                EXPLAIN_LEVEL     CHAR(1)     NOT NULL,
                                STMTNO            SMALLINT    NOT NULL,
                                SECTNO            SMALLINT    NOT NULL,
                                OPERATOR_ID       SMALLINT    NOT NULL,
                                OPERATOR_TYPE     CHAR(6)     NOT NULL,
                                TOTAL_COST        DOUBLE      NOT NULL,
                                IO_COST           DOUBLE      NOT NULL,
                                CPU_COST          DOUBLE      NOT NULL,
                                FIRST_ROW_COST    DOUBLE      NOT NULL,
                                RE_TOTAL_COST     DOUBLE      NOT NULL,
                                RE_IO_COST        DOUBLE      NOT NULL,
                                RE_CPU_COST       DOUBLE      NOT NULL,
                                COMM_COST         DOUBLE      NOT NULL,
                                FIRST_COMM_COST   DOUBLE      NOT NULL,
                                NODES_USED        CLOB(64K),
                                     FOREIGN KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA,
                                                  EXPLAIN_LEVEL,
                                                  STMTNO,
                                                  SECTNO)
                                     REFERENCES EXPLAIN_STATEMENT
                                     ON DELETE CASCADE )
```

EXPLAIN_PREDICATE Table Definition

```
CREATE TABLE EXPLAIN_PREDICATE ( EXPLAIN_REQUESTER CHAR(8)     NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP   NOT NULL,
                                 SOURCE_NAME       CHAR(8)     NOT NULL,
                                 SOURCE_SCHEMA     CHAR(8)     NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)     NOT NULL,
                                 STMTNO            SMALLINT    NOT NULL,
                                 SECTNO            SMALLINT    NOT NULL,
                                 OPERATOR_ID       SMALLINT    NOT NULL,
                                 PREDICATE_ID      SMALLINT    NOT NULL,
                                 HOW_APPLIED       CHAR(5)     NOT NULL,
                                 WHEN_EVALUATED    CHAR(3)     NOT NULL,
                                 RELOP_TYPE        CHAR(2)     NOT NULL,
                                 SUBQUERY          CHAR(1)     NOT NULL,
                                 FILTER_FACTOR     DOUBLE      NOT NULL,
                                 PREDICATE_TEXT    CLOB(64K),
                                      FOREIGN KEY (EXPLAIN_REQUESTER,
                                                   EXPLAIN_TIME,
                                                   SOURCE_NAME,
                                                   SOURCE_SCHEMA,
                                                   EXPLAIN_LEVEL,
                                                   STMTNO,
                                                   SECTNO)
                                      REFERENCES EXPLAIN_STATEMENT
                                      ON DELETE CASCADE )
```

EXPLAIN_STATEMENT Table Definition

```
CREATE TABLE EXPLAIN_STATEMENT ( EXPLAIN_REQUESTER CHAR(8)     NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP   NOT NULL,
                                 SOURCE_NAME       CHAR(8)     NOT NULL,
                                 SOURCE_SCHEMA     CHAR(8)     NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)     NOT NULL,
                                 STMTNO            SMALLINT    NOT NULL,
                                 SECTNO            SMALLINT    NOT NULL,
                                 QUERYNO           INTEGER     NOT NULL,
                                 QUERYTAG          CHAR(20)    NOT NULL,
                                 STATEMENT_TYPE    CHAR(2)     NOT NULL,
                                 UPDATABLE         CHAR(1)     NOT NULL,
                                 DELETABLE         CHAR(1)     NOT NULL
                                 TOTAL_COST        DOUBLE      NOT NULL,
                                 STATEMENT_TEXT    CLOB(64K)   NOT NULL,
                                 SNAPSHOT          BLOB(10M),
                                 QUERY_DEGREE      INTEGER     NOT NULL,
                                     PRIMARY KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA,
                                                  EXPLAIN_LEVEL,
                                                  STMTNO,
                                                  SECTNO),
                                     FOREIGN KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA)
                                     REFERENCES EXPLAIN_INSTANCE
                                     ON DELETE CASCADE )
```

EXPLAIN_STREAM Table Definition

```
CREATE TABLE EXPLAIN_STREAM ( EXPLAIN_REQUESTER CHAR(8)     NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP   NOT NULL,
                              SOURCE_NAME       CHAR(8)     NOT NULL,
                              SOURCE_SCHEMA     CHAR(8)     NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)     NOT NULL,
                              STMTNO            SMALLINT    NOT NULL,
                              SECTNO            SMALLINT    NOT NULL,
                              STREAM_ID         SMALLINT    NOT NULL,
                              SOURCE_TYPE       CHAR(1)     NOT NULL,
                              SOURCE_ID         SMALLINT    NOT NULL,
                              TARGET_TYPE       CHAR(1)     NOT NULL,
                              TARGET_ID         SMALLINT    NOT NULL,
                              OBJECT_SCHEMA     CHAR(8),
                              OBJECT_NAME       VARCHAR(18),
                              STREAM_COUNT      DOUBLE      NOT NULL,
                              COLUMN_COUNT      SMALLINT    NOT NULL,
                              PREDICATE_ID      SMALLINT    NOT NULL,
                              COLUMN_NAMES      CLOB(64K),
                              PMID              SMALLINT    NOT NULL,
                              SINGLE_NODE       CHAR(5),
                              PARTITION_COLUMNS CLOB(64K),
                                  FOREIGN KEY (EXPLAIN_REQUESTER,
                                               EXPLAIN_TIME,
                                               SOURCE_NAME,
                                               SOURCE_SCHEMA,
                                               EXPLAIN_LEVEL,
                                               STMTNO,
                                               SECTNO)
                                  REFERENCES EXPLAIN_STATEMENT
                                  ON DELETE CASCADE )
```

Appendix L. SQL Explain Tools (db2expln and dynexpln)

The **db2expln** tool describes the access plan selected for static SQL statements in the packages stored in the system catalog tables. It can be used to obtain a quick explanation of the chosen access plan for packages for which explain data was not captured at bind time.

The **dynexpln** tool describes the access plan selected for dynamic statements. It creates a static package for the statements and then uses the **db2expln** tool to describe them.

Note: Using this method of analysis is not as accurate as using the Explain Facility.

You can use these Explain tools to understand the access plan chosen for a particular SQL statement. Or, you could use the integrated Explain Facility (Chapter 13, "SQL Explain Facility" on page 377) in conjunction with Visual Explain to understand the access plan chosen for a particular SQL statement. Both dynamic and static SQL statements can be explained using the Explain Facility. One difference from the Explain tools is that with Visual Explain the Explain information is presented in a graphical format. Otherwise the level of detail provided in the two methods is equivalent.

To fully use the output of `db2expln`, and `dynexpln` you must understand:

- The different SQL statements supported and the terminology related to those statements (such as predicates in a SELECT statement).
- The purpose of a package (access plan). (See "Data Access Concepts and Optimization" on page 349 for this information.)
- The purpose and contents of the system catalog tables. (See Appendix I, "Catalog Views" on page 697 for this information.)
- Other concepts described in Part 3, "Tuning Application Performance" on page 263.

The following topics provide information about `db2expln` and `dynexpln`:

- Running db2expln and dynexpln
- Description of db2expln and dynexpln Output
- Examples of db2expln and dynxpln Output

Running db2expln and dynexpln

The explain tools (`db2expln` and `dynexpln`) are located in the `misc` subdirectory of your instance `sqllib` directory. If `db2expln` and `dynexpln` are not in your current directory, they must be in a directory that appears in your PATH environment variable.

The `db2expln` program connects and binds itself to a database using the `db2expln.bnd` file the first time the database is accessed. The `db2expln.bnd` file is in the `bnd` subdirectory of your `sqllib` directory.

To run db2expln, you must have SELECT privilege to the system catalog views as well as EXECUTE authority for the db2expln package. To run dynexpln, you must have BINDADD authority for the database as well as any privileges needed for the SQL statements being explained. (Note that if you have SYSADM or DBADM authority, you will automatically have all these authorization levels.)

Syntax for db2expln

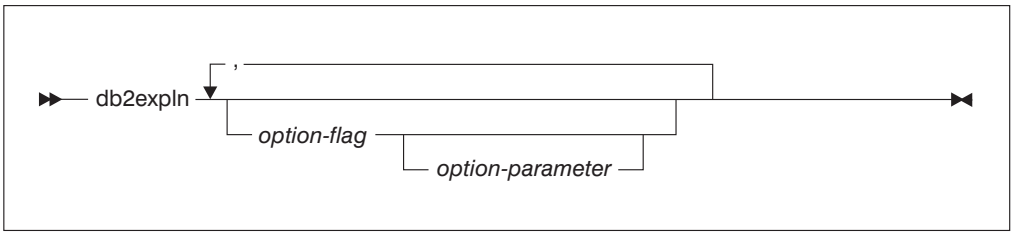


Figure 71. Syntax for db2expln

The table below summarizes the **option-flags** and **option-parameters** that may be used with the db2expln command. You can specify these options in any sequence and combination.

Table 99 (Page 1 of 2). db2expln Command Options

Option Flag	Option Parameter	Description
-c	creator	The user ID of the creator of the package. If you do not specify this option you will be prompted to provide it. You may specify the creator name using the pattern matching characters, percent sign (%) and underscore (_) that may be used in a LIKE predicate.
-d	database name	The name of the database which contains the packages to be explained. If you do not specify this option you will be prompted to provide it.
-e	escape character	Used to specify the character which is to be interpreted as an escape character rather than a pattern matching character. For example, the db2expln command to explain the package TESTID.CALC% is db2expln -c TESTID -p CALC%. However, this command would also explain any other plans that start with CALC. To explain just the TESTID.CALC% package, you must use an escape character. By changing the command to read: db2expln -c TESTID -e ! -p CALC!% you specify that the ! character will be used as an escape character and !% is interpreted as the % character.
-h or -?	(none)	Obtain help information about the input parameters. Specifying this option overrides all other options.
-i	(none)	Display operator IDs in the explained plan. The operator IDs allow the output from db2expln to be matched to the output from the Explain facility.

Table 99 (Page 2 of 2). db2expln Command Options

Option Flag	Option Parameter	Description
-l	(none)	The package name can be either lower or mixed-case if this option is specified. If this **-l** option is not specified, the package name is converted to uppercase
-o	output file	The name of the file to which `db2expln` will write the results. If you specify **-o** without a file name, you will be prompted for a file name. The default file name is `db2expln.out`.
-p	package name	The name of the package to be explained. If you do not specify this option you will be prompted to provide it. You may specify the package name using the pattern matching characters, percent sign (%) and underscore (_) that can be used in a LIKE predicate.
-s	section number	The section number to explain within the package. The number zero (0) may be specified if you wish to have all sections in the package explained. If the package creator (**-c**) or package name (**-p**) arguments imply that multiple packages will be explained, and thus multiple sections, the section value, if provided, is overridden with a zero (0). If you do not specify this option you will be prompted to provide it. Section numbers can be found by querying the system catalog SYSCAT.STATEMENTS (See Appendix I, “Catalog Views” on page 697 for a description of the system catalog tables.)
-t	(none)	The output is directed to the terminal. If you do not specify **-o** or **-t**, you will be prompted for a file name, with the default displaying the output at the terminal.
-u	user id and password	When connecting to a database, use the provided user ID and password. Both the user id and password must be valid according to naming conventions and be recognized by the database.

Some of the option flags in the above table may have special meaning to your Operating System and, as a result, may not be interpreted correctly in the `db2expln` command line. However, it may be possible to enter these characters by preceding them with an escape character. For more information, see your Operating System user's manual.

Help and initial status messages, produced by `db2expln`, are written to standard output. All prompts and other status messages produced by the explain tool are written to standard error. Explain text is written to standard output or to a file depending on the output option chosen.

With the **-p** and **-c** options, multiple plans can be explained with one invocation of explain by specifying string constants for packages and creators with LIKE patterns.

That is, the underscore (_) may be used to represent a single character, and the percent sign (%) may be used to represent the occurrence of zero or more characters.

For example, to explain all sections for all packages in a database named SAMPLE, with the results being written to the file **my.exp**, enter

```
db2expln -d SAMPLE -p % -c % -s 0 -o my.exp
```

Usage Notes for db2expln

The following are common messages displayed by `db2expln`:

- `No packages found for database <database>, package pattern: <creator>.<package>.`

  This message will appear in the output if no packages were found in the database that matched the specified pattern.
- `Bind messages can be found in db2expln.msg`

  This message will appear in the output if the bind of `db2expln.bnd` was not successful. Further information on the problems encountered will be found in the file `db2expln.msg` in the current directory.
- `Section number overridden to 0 for potential multiple packages.`

  This message will appear in the output if multiple packages may be encountered by `db2expln`. This action will be taken if one of the pattern matching characters is used in the package or creator input arguments.
- `No static sections qualify for database <database>, package <creator>.<package>, section <section>.`

  This message will appear in the output if the specified package only contains dynamic SQL statements which means that there are no static sections.
- `Database <database>, package <creator>.<package> is not valid. Rebind and then rerun db2expln.`

  This message will appear in the output if the package specified is currently not valid. As directed, reissue the BIND or REBIND command for the plan to re-create a valid package in the database, and then rerun `db2expln`.
- `Section not processed: Produced by unsupported release.`

  This message will also appear in the output if the section currently being processed was produced by a release of DB2 other than the one for which this `db2expln` executable was provided. In this case, use the copy of `db2expln` from the release of DB2 that produced the section.

SQL Statements Excluded: The following statements will not be explained:

- BEGIN/END DECLARE SECTION
- BEGIN/END COMPOUND
- INCLUDE
- WHENEVER
- COMMIT and ROLLBACK

- CONNECT
- OPEN cursor
- FETCH
- CLOSE cursor
- PREPARE
- EXECUTE
- EXECUTE IMMEDIATE
- DESCRIBE
- Dynamic DECLARE CURSOR

Each sub-statement within a compound SQL statement may have its own section, which can be explained by **db2expln**.

Syntax for dynexpln

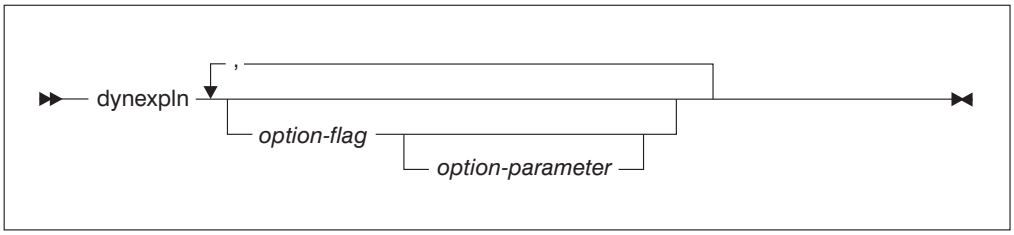


Figure 72. Syntax for dynexpln

The table below summarizes the **option-flags** and **option-parameters** that may be used with the `dynexpln` command. You can specify these options in any sequence and combination.

Table 100 (Page 1 of 2). dynexpln Command Options

Option Flag	Option Parameter	Description
-d	database name	The name of the database to use when explaining statements. If you do not specify this option you will be prompted to provide it.
-e	statement terminator	The character used to indicate that the end of an SQL statement has been reached. The default is that there is no statement terminator. If you use this option, `dynexpln` will use the specified termination character to separate the statements. If you do not use this option, each line of the file will be assumed to be a separate SQL statement.
-f	input file	The name of the file which contains the SQL statements to be explained. Unless you use the statement terminator (**-e**) option, only one SQL statement should appear on each line of the file. SQL comments may be entered into the file. An SQL comment starts with `--` and goes to the end of the line.

Table 100 (Page 2 of 2). dynexpln Command Options

Option Flag	Option Parameter	Description
-h or -?	(none)	Obtain help information about the input parameters. Specifying this option overrides all other options.
-i	(none)	Display operator IDs in the explained plan. The operator IDs allow the output from `dynexpln` to be matched to the output from the Explain facility.
-o	output file	The name of the file to which `dynexpln` will write the results.
-s	SQL statement	The SQL statement to be explained. If you do not specify this option and you do not specify the input file (**-f**) optional parameter, you will be prompted to provide the SQL statement to be explained. If you specify both this option and the input file (**-f**) optional parameter, `dynexpln` will first describe the statements provided by the SQL statement (**-s**) option and then describe the statements in the input file (**-f**).
-t	(none)	The output is directed to the terminal. If both the output (**-o**) and **-t** options are specified, then the output is directed to the terminal. If you do not specify the output file (**-o**) or **-t** options, you will be prompted for a file name, with the default displaying the output at the terminal.
-u	user id and password	When connecting to the database, use the provided user id and password. Both the user id and password must be valid according to naming conventions and be recognized by the database.

Some of the option flags in the above table may have special meaning to your Operating System and, as a result, may not be interpreted correctly in the `dynexpln` command line. However, it may be possible to enter these characters by preceding them with an escape character. For more information, see your Operating System user's manual.

If you use the statement terminator (**-e**) option, you may enter multiple statements using the SQL statement (**-s**) option. If you do this, you should separate the statements with the termination character.

Help and initial status messages, produced by `dynexpln`, are written to standard output. All prompts and other status messages produced by the explain tool are written to standard error. Explain text is written to standard output or to a file depending on the output option chosen.

For example, to connect to a database named SAMPLE and explain all the statements in the file `TRYIT`, with the results being written to the file `my.exp`, enter

```
dynexpln -d SAMPLE -f TRYIT -o my.exp
```

Usage Notes for dynexpln

To explain dynamic statements, dynexpln creates a static application for the statements and then invokes db2expln. To create the static statements, dynexpln generates a trivial C program with the statements and then calls the DB2 precompiler to create the package. (The generated C program is not complete and cannot be compiled; it only contains enough information that the precompiler can build the package.)

The following are common messages displayed by dynexpln:

- All error messages from db2expln.

  Since dynexpln invokes db2expln, it is possible to see most of db2expln's error messages.
- Error connecting to the database.

  This message will appear in the output if an error occurred connecting to the database. A CLI error message will also be displayed indicating why the connection could not be completed. Correct the cause of the error and run dynexpln again.
- The file "<filename>" must be removed before dynexpln will run.

  This message will appear if the given file exists at the time dynexpln is run. Remove the file or change the value of the DYNEXPLN_PACKAGE environment variable to change the name of the file which will be created and run dynexpln again.
- The package "<creator>.<package>" must be dropped before dynexpln will run.

  This message will appear if the given package exists at the time dynexpln is run. Drop the package and run or change the value of the **DYNEXPLN_PACKAGE** environment variable to change the name of the package which will be created and run dynexpln again.
- Error writing file "<filename>".

  This message will appear if the given file cannot be written to. Ensure that dynexpln can write files in the current directory and run it again.
- Error reading input file "<filename>".

  This message will appear if the file given with the **-f** option cannot be read from. Ensure that the file exists and that dynexpln can read it. Then run dynexpln again.

Environment Variables: There are two different environment variables that can be used in conjunction with dynexpln:

- **DYNEXPLN_OPTIONS** are the SQL precompiler options you use when building the package for your statements. Use the same syntax variable as you would when issuing a PREP command through CLP.

  For example: DYNEXPLN_OPTIONS="OPTLEVEL 5 BLOCKING ALL"
- **DYNEXPLN_PACKAGE** is the name of the package which is created in the database. The statements to be described are placed in this package. If this variable is not defined, the package is given a default value of **DYNEXPLN**. (Only the first eight characters of the name in this environment variable are used.)

The name is also used to create the names for the intermediate files that dynexpln uses.

Description of db2expln and dynexpln Output

In the output, the explain information for each package is broken into two parts:

- Package information such as date of bind and relevant bind options
- Section information such as the section number followed by the SQL statement being explained. Beneath the section information will be the explain output of the access plan chosen for the SQL statement shown.

The steps of an access plan, or section, will be presented in the order that the database manager executes them. Each major step will be shown as a left-justified heading with information about that step indented beneath it. The explain output for the access plan has indentation bars provided in the left margin of the output. These bars also provide the "scope" for the operation; operations at a lower (that is, further to the right) level of indentation within the same operation are processed before returning to the previous level of indentation.

It is important to remember that the access plan chosen was based on an augmented version of the original SQL statement (the one shown in the output). For example, the original statement may cause any number of triggers and constraints to be activated. As well, the SQL statement may be rewritten to an equivalent but more efficient format by the Query Rewrite component of the SQL Compiler. All of these factors are included in the information presented to the Optimizer when it determines the most efficient plan to satisfy the statement. Thus, the access plan shown in the explain output may differ substantially from the access plan that one might expect for the original SQL statement. The integrated Explain facility (see Chapter 13, “SQL Explain Facility” on page 377) shows the actual SQL statement used for optimization in the form of an SQL-like statement which is created by reverse-translating the internal representation of the query.

When comparing output from db2expln or dynexpln to the output of the Explain facility, the operator ID option (**-i**) can be very useful. Each time db2expln or dynexpln starts processing a new operator from the Explain facility, the operator ID number will be printed to the left of the explained plan. The operator IDs can be used to match up the steps in the different representations of the access plan. Note that there is not always a one-to-one correspondence between the operators in the Explain facility output and the operations shown by db2expln and dynexpln.

The following topics describe the explain text that may be produced by db2expln and dynexpln:

- Table Access
- Temporary Tables
- Joins
- Data Streams
- Insert, Update, and Delete
- Row Identifier (RID) Preparation

- Aggregation
- Parallel Processing.
- Miscellaneous Statements.

Table Access

This statement tells the name and type of table being accessed. It has two formats that are used:

1. Regular tables:

   ```
   Access Table Name = schema.name  ID = n
   ```

   where:

   - *schema.name* is the fully-qualified name of the table being accessed
   - *ID* is the corresponding TABLEID column in the SYSCAT.TABLES catalog for a table

2. Temporary tables:

   ```
   Access Temp ID = tn
   ```

   where:

   - *ID* is the corresponding identifier assigned by `db2expln`

Following the table access statement, additional statements will be provided to further describe the access. These statements will be indented under the table access statement. The possible statements are:

- Number of Columns
- Parallel Scan
- Scan Direction
- Row Access Method
- Lock Intents
- Predicates
- Miscellaneous Table Statements.

Number of Columns

The following statement indicates the number of columns being used from each row of the table:

```
#Columns = n
```

Parallel Scan

The following statement indicates that the database manager will use several subagents to read from the table in parallel:

```
Parallel Scan
```

If this text is not shown, the table will only be read from by one agent (or subagent).

Scan Direction

The following statement indicates that the database manager will read rows in a reverse order:

```
Scan Direction = Reverse
```

If this text is not shown, the scan direction is forward, which is the default. Note that an index scan can only read data in a forward order.

Row Access Method

One of the following statements will be displayed, indicating how the qualifying rows in the table are being accessed:

- The `Relation Scan` statement indicates that the table is being sequentially scanned to find the qualifying rows.
  - The following statement indicates that no prefetching of data will be done:

    ```
    Relation Scan
    |  Prefetch: None
    ```

  - The following statement indicates that the optimizer has predetermined the number of pages that will be prefetched:

    ```
    Relation Scan
    |  Prefetch: n Pages
    ```

  - The following statement indicates that data should be prefetched:

    ```
    Relation Scan
    |  Prefetch: Eligible
    ```

  - The following statement indicates that the qualifying rows are being identified and accessed through an index:

    ```
    Index Scan:  Name = schema.name  ID = xx
    ```

    where:
    - *schema.name* is the fully-qualified name of the index being scanned
    - *ID* is the corresponding IID column in the SYSCAT.INDEXES catalog view.

    The following statements are provided to clarify the type of index scan:
    - `#Key Columns = n`

      This statement shows the number of range-delimiting predicates, that is, the number of columns in the index key (from left to right) being used to delimit the index scan range. If `#Key Columns` equals zero, a full scan of the index is being performed. This might be done if the database manager decides that an index scan is cheaper than a relation scan for evaluating some of the predicates, or if the index is just being used to order the output.
    - If there are more predicates specifying where to start scanning the index than there are predicates specifying where to stop scanning the index, then the following statement will appear:

```
#Start Keys = n1
#Stop Keys = n2
```

- `Index-only Access`

  If all the needed columns can be obtained from the index key, this statement will appear and no table data will be accessed.
- The following statement indicates that no prefetching of index pages will be done:

  ```
  Index Prefetch: None
  ```
- The following statement indicates that index pages should be prefetched:

  ```
  Index Prefetch: Eligible
  ```
- The following statement indicates that no prefetching of data pages will be done:

  ```
  Data Prefetch: None
  ```
- The following statement indicates that data pages should be prefetched:

  ```
  Data Prefetch: Eligible
  ```
- If there are predicates that can be passed to the Index Manager to help qualify index entries, the following statement is used to show the number of predicates:

  ```
  Sargable Index Predicate(s)
  |  #Predicates = n
  ```

– The `Fetch Direct` statement indicates that the qualifying rows are being accessed by using row IDs (RIDs) that were prepared earlier in the access plan.

Lock Intents

For each table access, the type of lock that will be acquired at the table and row levels is shown with the following statement:

```
Lock Intents
|  Table: xxxx
|  Row  : xxxx
```

Possible values for a table lock are:

- Exclusive
- Intent Exclusive
- Intent None
- Intent Share
- Share
- Share Intent Exclusive
- Super Exclusive
- Update

Possible values for a row lock are:

- Exclusive

- Next Key Exclusive (does not appear in `db2expln` output)
- None
- Share
- Next Key Share
- Update

The explanation of these lock types is found in “Attributes of Locks” on page 271.

Predicates

There are two statements that provide information about the predicates used in an access plan:

1. The following statement indicates the number of predicates that will be evaluated once the data has been returned:

   ```
   Residual Predicate(s)
   |  #Predicates = n
   ```

2. The following statement indicates the number of predicates that will be evaluated while the data is being accessed. The count of predicates does not include push-down operations such as aggregation or sort.

   ```
   Sargable Predicate(s)
   |  #Predicates = n
   ```

The number of predicates shown in the above statements may not reflect the number of predicates provided in the SQL statement because predicates can be:

- Applied more than once within the same query
- Transformed and extended with the addition of implicit predicates during the query optimization process
- Transformed and condensed into fewer predicates during the query optimization process.

Miscellaneous Table Statements

- The following statement indicates that only one row will be accessed:

  ```
  Single Record
  ```

- The following statement appears when the isolation level used for this table access uses a different isolation level than the package:

  ```
  Isolation Level: xxxx
  ```

  A different isolation level may be used for a number of reasons, including:
  - A package was bound with Repeatable Read and affects Referential Integrity constraints; the access of the parent table to check referential integrity constraints is downgraded to an isolation level of Cursor Stability to avoid holding unnecessary locks on this table.
  - A package bound with Uncommitted Read issues a DELETE or UPDATE statement; the table access for the actual delete is upgraded to Cursor Stability.

Temporary Tables

A temporary table is used by an access plan to store data during its execution in a transient or temporary work table. This table only exists while the access plan is being executed. Generally, temporary tables are used when subqueries need to be evaluated early in the access plan, or when intermediate results will not fit in the available memory.

If a temporary table needs to be created, then one of two possible statements may appear. These statements indicate that a temporary table is to be created and rows inserted into it. The ID is an identifier assigned by `db2expln` for convenience when referring to the temporary table. This ID is prefixed with the letter 't' to indicate that the table is a temporary table.

1. The following statement indicates an ordinary temporary table will be created:

   ```
   Insert Into Temp Table  ID = tn
   ```

2. The following statement indicates an ordinary temporary table will be created by multiple subagents in parallel:

   ```
   Insert Into Shared Temp Table  ID = tn
   ```

3. The following statement indicates a sorted temporary table will be created:

   ```
   Insert Into Sorted Temp Table  ID = tn
   ```

4. The following statement indicates a sorted temporary table will be created by multiple subagents in parallel:

   ```
   Insert Into Sorted Shared Temp Table  ID = tn
   ```

Each of the above statements will be followed by:

```
#Columns = n
```

which indicates how many columns are in each row being inserted into the temporary table.

Sorted Temporary Tables

Sorted temporary tables can result from such operations as:

- ORDER BY
- DISTINCT
- GROUP BY
- Merge Join
- '= ANY' subquery
- '<> ALL' subquery
- INTERSECT or EXCEPT
- UNION (without the ALL keyword)

A number of additional statements may follow the original creation statement for a sorted temporary table:

- The following statement indicates the number of key columns used in the sort:

```
#Sort Key Columns = n
```

The specific columns used in the sort are not listed. However, you can deduce them from the source SQL statement.

- The following statements provide estimates of the number of rows and the row size so that the optimal sort heap can be allocated at run time.

```
Sortheap Allocation Parameters:
  #Rows     = n
  Row Width = n
```

- For sorts in a symmetric multi-processor (SMP) environment, the type of sort to be performed is indicated by one of the following statements:

```
Use Partitioned Sort
Use Shared Sort
Use Replicated Sort
Use Round-Robin Sort
```

For a description of the different sorting techniques, see “Sorting” on page 412.

- The following statements indicate whether or not the result from the sort will be left in the sort heap:

```
Piped
```

and

```
Not Piped
```

If a piped sort is indicated, the database manager will keep the sorted output in memory, rather than placing the sorted result in another temporary table. (For a description of piped versus non-piped sorts, see “Influence of Sorting on the Optimizer” on page 372.)

- The following statement indicates that duplicate values will be removed during the sort:

```
Duplicate Elimination
```

- If aggregation is being performed in the sort, it will be indicated by one of the following statements:

```
Partial Aggregation
Intermediate Aggregation
```

Temporary Table Completion

After a table access that contains a push-down operation to create a temporary table (that is, a create temporary table that occurs within the scope of a table access), there will be a "completion" statement, which handles end-of-file by getting the temporary table ready to provide rows to subsequent temporary table access. One of the following lines will be displayed:

```
Temp Table Completion  ID = tn
Shared Temp Table Completion  ID = tn
Sorted Temp Table Completion  ID = tn
Sorted Shared Temp Table Completion  ID = tn
```

Joins

There are two types of joins (see “Join Concepts” on page 359 for a description of these joins):

- Merge join
- Nested loop join.

When the time comes in the execution of a section for a join to be performed, one of the following statements is displayed:

```
Merge Join
```

or

```
Nested Loop Join
```

It is possible for a left outer join to be performed. A left outer join is indicated by one of the following statements:

```
Left Outer Merge Join
```

or

```
Left Outer Nested Loop Join
```

The outer table of the join will be the table referenced in the previous access statement shown in the output. The inner table of the join will be the table referenced in the access statement that is contained within the scope of the join statement.

For a merge join, the following additional statements may appear:

- In some circumstances, a join simply needs to determine if any row in the inner table matches the current row in the outer. This is indicated with the statement:

  ```
  Early Out: Single Match Per Outer Row
  ```

- It is possible to apply predicates after the join has completed. The number of predicates being applied will be indicated as follows:

  ```
  Residual Predicate(s)
  |  #Predicates = n
  ```

For a nested loop join, the following additional statement may appear immediately after the join statement:

```
Piped Inner
```

This statement indicates that the inner table of the join is the result of another series of operations. This is also referred to as a *composite inner*.

If a join involves more than two tables, the explain steps should be read from top to bottom. For example, suppose the explain output has the following flow:

```
Access ..... W
Join
|  Access ..... X
Join
|  Access ..... Y
Join
|  Access ..... Z
```

The steps of execution would be:

1. Take a row that qualifies from W
2. Join row from W with (next) row from X and call the result P1 (for partial join result number 1)
3. Join P1 with (next) row from Y to create P2
4. Join P2 with (next) row from Z to obtain one complete result row
5. If there are more rows in Z, go to step 4
6. If there are more rows in Y, go to step 3
7. If there are more rows in X, go to step 2
8. If there are more rows in W, go to step 1.

Data Streams

Within an access plan, there is often a need to control the creation and flow of data from one series of operations to another. The data stream concept allows a group of operations within an access plan to be controlled as a unit. The start of a data stream is indicated by the following statement:

```
Data Stream n
```

where n is a unique identifier assigned by db2expln for ease of reference. The end of a data stream is indicated by:

```
End of Data Stream n
```

All operations between these statements are considered part of the same data stream.

A data stream has a number of characteristics and one or more statements can follow the initial data stream statement to describe these characteristics:

- The following statements indicate when and how the data stream is created:

  ```
  Evaluate at Open
  Evaluate at Application
  Forced Evaluate at Application
  ```

  The data stream is either fully created once when it is first opened (Evaluate at Open) or each time it is accessed (Evaluate at Application). If the data stream is evaluated at application, it can be forced to be fully evaluated with each access or it can be allowed to be evaluated as required by the particular access.

- Similar to a sorted temporary table, the following statements indicate whether or not the results of the data stream will be kept in memory:

  ```
  Piped
  ```

and

```
Not Piped
```

As was the case with temporary tables, a piped data stream may be written to disk, if insufficient memory exists at execution time. The access plan will provide for both possibilities.

- The following statement indicates that only a single record is required from this data stream:

  ```
  Single Record
  ```

When a data stream is accessed, the following statement will appear in the output:

```
Access Data Stream n
```

Insert, Update, and Delete

The explain text for these SQL statements is self-explanatory. Possible statement text for these SQL operations can be:

- `Insert: Table Name = schema.name`
- `Update: Table Name = schema.name`
- `Delete: Table Name = schema.name`

Row Identifier (RID) Preparation

For some access plans, it is more efficient if the qualifying row identifiers (RIDs) are sorted and duplicates removed (in the case of index ORing) or that a technique is used to identify RIDs appearing in all indexes being accessed (in the case of index ANDing) before the actual table access is performed. There are three main uses of RID preparation as indicated by the explain statements:

- The following statement indicates that “Index ORing” is used to prepare the list of qualifying RIDs:

  ```
  Index ORing RID Preparation
  ```

  Index ORing refers to the technique of making more than one index access and combining the results to include the distinct RIDs that appear in any of the indexes accessed. The optimizer will consider index ORing when predicates are connected by OR keywords or there is an IN predicate. The index accesses can be on the same index or different indexes.
- Another use of RID preparation is to prepare the input data to be used during list prefetch, as indicated by the following:

  ```
  List Prefetch RID Preparation
  ```

- The following statement indicates that “Index ANDing” is used to prepare the list of qualifying RIDs:

  ```
  Index ANDing RID Preparation
  ```

  Index ANDing refers to the technique of making more than one index access and combining the results to include RIDs that appear in all of the indexes accessed. Index ANDing processing is started with the statement:

```
Index ANDing
```

If the optimizer has estimated the size of the result set, the estimate is shown with the following statement:

```
Optimizer Estimate of Set Size: n
```

Index ANDing filter operations process RIDs and use bit filter techniques to determine the RIDs which appear in every index accessed. The following statements indicate that RIDs are being processed for index ANDing:

```
Index ANDing Bitmap Build
Index ANDing Bitmap Probe
Index ANDing Bitmap Build and Probe
```

If the optimizer has estimated the size of the result set for a bitmap, the estimate is shown with the following statement:

```
Optimizer Estimate of Set Size: n
```

For any type of RID preparation, if list prefect can be performed it will be indicated with the statement:

```
Prefetch: Enabled
```

Aggregation

Aggregation is performed on those rows meeting the specified criteria, if any, provided by the SQL statement predicates. If some sort of aggregate function is to be done, one of the following statements appears:

```
Aggregation
Predicate Aggregation
Partial Aggregation
Partial Predicate Aggregation
Intermediate Aggregation
Intermediate Predicate Aggregation
Final Aggregation
Final Predicate Aggregation
```

Predicate aggregation states that the aggregation operation has been pushed-down to be processed as a predicate when the data is actually accessed.

Beneath either of the above aggregation statements will be a indication of the type of aggregate function being performed:

- `Group By`
- `Column Function(s)`
- `Single Record`

The specific column function can be derived from the original SQL statement. A single record is fetched from an index to satisfy a MIN or MAX operation.

If predicate aggregation is used, then subsequent to the table access statement in which the aggregation appeared, there will be an aggregation "completion", which

carries out any needed processing on completion of each group or on end-of-file. One of the following lines is displayed:

```
Aggregation Completion
Partial Aggregation Completion
Intermediate Aggregation Completion
Final Aggregation Completion
```

Parallel Processing

Executing an SQL statement in parallel (using either intra-partition parallel or partition parallel) requires some special operations. The operations for parallel plans are described below.

- When running an intra-partition parallel plan, portions of the plan will be executed simultaneously using several subagents. The creation of the subagents is indicated by the statement:

  ```
  Process Using n Subagents
  ```

- When running a partition parallel plan, the section is broken into several subsections. Each subsection is sent to one or more nodes to be run. An important subsection is the *coordinator subsection*. The coordinator subsection is the first subsection in every plan. It gets control first and is responsible for distributing the other subsections and returning results to the calling application.

  The distribution of subsections is indicated by the statement:

  ```
  Distribute Subsection #n
  ```

  The details of how a subsection is distributed:

  – Under certain circumstances, it is possible for a subsection that would normally be sent to the coordinator node to be executed directly by the coordinator subsection. If this is potentially possible, it will be indicated by:

    ```
    Locally Bypassable
    ```

  – The nodes that receive a subsection can be determined in one of seven ways:

    ```
    Directed by Hash
    |  #Columns = n
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

    This indicates that the subsection will be sent to a node within the nodegroup based on the value of the columns.

    ```
    Directed by Node Number
    ```

    This indicates that the subsection will be sent to a predetermined node. (This is frequently seen when the statement uses the NODENUMBER() function.)

    ```
    Directed by Partition Number
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

    This indicates that the subsection will be sent to the node corresponding to a predetermined partition number in the given nodegroup. (This is frequently seen when the statement uses the PARTITION() function.)

    ```
    Directed by Position
    ```

This indicates that the subsection will be sent to the node that provided the current row for the application's cursor.

```
Directed to Single Node
|  Node Number = n
```

This indicates that only one node, determined when the statement was compiled, will receive the subsection.

```
Directed to Coordinator Node
```

The subsection will be executed on the coordinator node.

```
Broadcast to Node List
|  Nodes = n1, n2, n3, ...
```

This indicates that the subsection will be sent to all the nodes listed.

- Table queues are used to move data between subsections in a partitioned database environment or between subagents in a symmetric multi-processor (SMP) environment. Table queues are described as follows:
  - The following statements indicate that data is being inserted into a table queue:

    ```
    Insert Into Synchronous Table Queue  ID = qn
    Insert Into Asynchronous Table Queue  ID = qn
    Insert Into Synchronous Local Table Queue  ID = qn
    Insert Into Asynchronous Local Table Queue  ID = qn
    ```

  - For database partition table queues, the destination for rows inserted into the table queue is described by one of the following:

    ```
    Broadcast to Coordinator Node
    ```

    All rows are sent to the coordinator node.

    ```
    Broadcast to All Nodes of Subsection n
    ```

    All rows are sent to every database partition that the given subsection is running on.

    ```
    Hash to Specific Node
    ```

    Each row is sent to a database partition based on the values in the row.

    ```
    Send to Specific Node
    ```

    Each row is sent to a database partition determined while the statement is executing.

    ```
    Send to Random Node
    ```

    Each row is sent to a random database partition.
  - In some situations, a database partition table queue will have to temporarily overflow some rows to a temporary table. This possibility is identified by the statement:

    ```
    Rows Can Overflow to Temporary Table
    ```

  - The following statements indicate that data is being retrieved from a table queue:

```
Access Table Queue  ID = qn
Access Local Table Queue  ID = qn
```

These messages are always followed by an indication of the number of columns being retrieved.

```
#Columns = n
```

– If the table queue sorts the rows at the receiving end, the table queue access will also have one of the following messages:

```
Output Sorted
Output Sorted and Unique
```

These messages are followed by an indication of the number of keys used for the sort operation.

```
#Key Columns = n
```

– If predicates will be applied to rows by the receiving end of the table queue, the following message is shown:

```
Residual Predicate(s)
|  #Predicates = n
```

- Some subsections in a partitioned database environment explicitly loop back to the start of the subsection with the statement:

```
Jump Back to Start of Subsection
```

Miscellaneous Statements

- Sections for data definition language statements will be indicated in the output with the following:

```
DDL Statement
```

No additional explain output is provided for DDL statements.

- Sections for SET statements for the updatable special registers such as **CURRENT EXPLAIN SNAPSHOT** will be indicated in the output with the following:

```
SET Statement
```

No additional explain output is provided for SET statements.

- If the SQL statement contains the DISTINCT clause, the following text may appear in the output:

```
Distinct Filter  #Columns = n
```

where `n` is the number of columns involved in obtaining distinct rows. To retrieve distinct row values, the rows must be ordered so that duplicates can be skipped. This statement will not appear if the database manager does not have to explicitly eliminate duplicates, as in the following cases:

– A unique index exists and all the columns in the index key are part of the DISTINCT operation
– Duplicates that can be eliminated during sorting.

- The following statement will appear if the next operation is dependent on a specific record identifier:

```
Positioned Operation
```

This statement would appear for any SQL statement that uses the WHERE CURRENT OF syntax.

- The following statement will appear if there are predicates that must be applied to the result but that could not be applied as part of another operation:

```
Residual Predicate Application
|  #Predicates = n
```

- The following statement will appear if there is a UNION operator in the SQL statement:

```
UNION
```

- The following statement will appear if there is an operation in the access plan, whose sole purpose is to produce row values for use by subsequent operations:

```
Table Constructor
|  n-Row(s)
```

  Table constructors can be used for transforming values in a set into a series of rows that are then passed to subsequent operations. When a table constructor is prompted for the next row, the following statement will appear:

```
Access Table Constructor
```

- The following statement will appear if there is an operation which is only processed under certain conditions:

```
Conditional Evaluation
|  Condition #n:
|  |  #Predicates = n
|  Action #n:
```

  Conditional evaluation is used to implement such activities as the SQL CASE statement or internal mechanisms such as referential integrity constraints or triggers. If no action is shown, then only data manipulation operations are processed when the condition is true.
- One of the following statements will appear if an ALL, ANY, or EXISTS subquery is being processed in the access plan:
  - `ANY/ALL Subquery`
  - `EXISTS Subquery`
  - `EXISTS SINGLE Subquery`
- The following statement will appear if there are rows being returned to the application:

```
Return Data to Application
|  #Columns = n
```

Examples of db2expln and dynxpln Output

Three examples are shown here to help understand the layout and format of the output from `db2expln` and `dynexpln`. These examples were run against the SAMPLE database as provided with DB2. A brief discussion is provided for each example. Significant differences from one example to the next have been shown in **bold**.

Example One: "No Parallelism" Plan

This example is simply requesting a list of all employee names, their jobs, department name and location, and the project name(s) on which they are working. The essence of this access plan is that merge joins are used to join the relevant data from each of the specified tables. Since no indexes are available, the access plan does a relation scan of each table with a pushed-down sort operation inside each table access.

```
****************** PACKAGE ***************************************

Package Name = QUERY.EXAMPLE
     Prep Date = 1997/04/07
     Prep Time = 09:21:27:087

     Bind Timestamp = 1997-04-07-09.21.27.879168

     Isolation Level          = Cursor Stability
     Blocking                 = Block Unambiguous Cursors
     Query Optimization Class = 5

     Partition Parallel       = No
     Intra-Partition Parallel = No

     Function Path            = "SYSIBM", "SYSFUN", "QUERY"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT X.LASTNAME, X.JOB, Y.DEPTNAME, Y.LOCATION, Z.PROJNAME
  FROM EMPLOYEE X, DEPARTMENT Y, PROJECT Z
  WHERE X.WORKDEPT = Y.DEPTNO AND X.WORKDEPT = Z.DEPTNO AND Y.DEPTNO
          = Z.DEPTNO


Estimated Cost        = 373066
Estimated Cardinality = 1600000

Access Table Name = QUERY.EMPLOYEE  ID = 5
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Share
|  Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 3
|  |  #Sort Key Columns = 1
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 1000
|  |  |  Row Width = 36
```

```
|  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
Merge Join
|  Access Table Name = QUERY.DEPARTMENT  ID = 4
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Share
|  |  Insert Into Sorted Temp Table  ID = t2
|  |  |  #Columns = 3
|  |  |  #Sort Key Columns = 1
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 1000
|  |  |  |  Row Width = 60
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
Merge Join
|  Access Table Name = QUERY.PROJECT  ID = 7
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Share
|  |  Insert Into Sorted Temp Table  ID = t3
|  |  |  #Columns = 2
|  |  |  #Sort Key Columns = 1
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 1000
|  |  |  |  Row Width = 36
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t3
|  Access Temp Table  ID = t3
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
Return Data to Application
|  #Columns = 5

End of section
```

The first part of the plan accesses the EMPLOYEE table to get the last name, job, and department number. This information is placed in a sorted temporary table since merge joins require their input to be sorted. The resulting temporary table is the outer table of the first merge join operator.

Note that even though the sorted temporary table indicates that the result will be piped (that is, kept in memory), the access plan treats this case as if the result was written to a real table and provides a scan of the table to send the result to the user. This allows the access plan to be independent of the volume of data present at execution time. The temporary table accessed at execution time will either reside in memory or on disk.

The inner of this join operator is another sorted temporary table which is also piped. This temporary table is created from the DEPARTMENT table and contains the department name and location. In a similar fashion, the second merge join accesses and sorts the information needed from the PROJECT table.

Example Two: Non-Partitioned Parallel Plan

This example shows the same SQL statement as “Example One: "No Parallelism" Plan” on page 805, but this query has been compiled for a 4-way SMP machine.

```
******************** PACKAGE ************************************

Package Name = QUERY.EXAMPLE
       Prep Date = 1997/04/07
       Prep Time = 09:23:04:018

       Bind Timestamp = 1997-04-07-09.23.04.182277

       Isolation Level          = Cursor Stability
       Blocking                 = Block Unambiguous Cursors
       Query Optimization Class = 5

       Partition Parallel       = No
       Intra-Partition Parallel = Yes (Bind Degree = 4)

       Function Path            = "SYSIBM", "SYSFUN", "QUERY"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT X.LASTNAME, X.JOB, Y.DEPTNAME, Y.LOCATION, Z.PROJNAME
  FROM EMPLOYEE X, DEPARTMENT Y, PROJECT Z
  WHERE X.WORKDEPT = Y.DEPTNO AND X.WORKDEPT = Z.DEPTNO AND Y.DEPTNO
          = Z.DEPTNO

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 373066
```

```
Estimated Cardinality = 1600000

Process Using 4 Subagents
|  Access Table Name = QUERY.EMPLOYEE  ID = 5
|  |  #Columns = 3
|  |  Parallel Scan
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Share
|  |  Insert Into Sorted Shared Temp Table  ID = t1
|  |  |  #Columns = 3
|  |  |  #Sort Key Columns = 1
|  |  |  Use Partitioned Sort
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 1000
|  |  |  |  Row Width = 36
|  |  |  Piped
|  Sorted Shared Temp Table Completion  ID = t1
|  Access Temp Table  ID = t1
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Merge Join
|  |  Access Table Name = QUERY.DEPARTMENT  ID = 4
|  |  |  #Columns = 3
|  |  |  Parallel Scan
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Share
|  |  |  Insert Into Sorted Shared Temp Table  ID = t2
|  |  |  |  #Columns = 3
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  Use Partitioned Sort
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 1000
|  |  |  |  |  Row Width = 60
|  |  |  |  Piped
|  |  Sorted Shared Temp Table Completion  ID = t2
|  |  Access Temp Table  ID = t2
|  |  |  #Columns = 3
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  Merge Join
|  |  Access Table Name = QUERY.PROJECT  ID = 7
|  |  |  #Columns = 2
|  |  |  Parallel Scan
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
```

```
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Share
|  |  |  Insert Into Sorted Shared Temp Table  ID = t3
|  |  |  |  #Columns = 2
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  Use Replicated Sort
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 1000
|  |  |  |  |  Row Width = 36
|  |  |  |  Piped
|  |  Sorted Shared Temp Table Completion  ID = t3
|  |  Access Temp Table  ID = t3
|  |  |  #Columns = 2
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 5
Return Data to Application
|  #Columns = 5

End of section
```

This plan is almost identical to the plan in the first example. The main differences are the creation of four subagents when the plan first starts and the table queue at the end of the plan to gather the results of each of subagent's work before returning them to the application.

Example Three: Partitioned Database Plan

This example shows the same SQL statement as “Example One: "No Parallelism" Plan” on page 805, but this query has been compiled on a partitioned database made up of three database partitions. Additionally, `db2expln` was run with the **-i** option so the Explain facility operator IDs appear in parenthesis down the left side of the plan.

```
****************** PACKAGE ************************************

Package Name = QUERY.EXAMPLE
       Prep Date = 1997/04/07
       Prep Time = 09:51:24:076

       Bind Timestamp = 1997-04-07-09.51.24.765882

       Isolation Level          = Cursor Stability
       Blocking                 = Block Unambiguous Cursors
       Query Optimization Class = 5

       Partition Parallel       = Yes
       Intra-Partition Parallel = No

       Function Path            = "SYSIBM", "SYSFUN", "QUERY"

------------------- SECTION -------------------------------------
```

```
Section = 1


SQL Statement:

  SELECT X.LASTNAME, X.JOB, Y.DEPTNAME, Y.LOCATION, Z.PROJNAME
  FROM EMPLOYEE X, DEPARTMENT Y, PROJECT Z
  WHERE X.WORKDEPT = Y.DEPTNO AND X.WORKDEPT = Z.DEPTNO AND Y.DEPTNO
          = Z.DEPTNO

Buffered Insert          = No

Estimated Cost        = 1766063
Estimated Cardinality = 43200000

        Coordinator Subsection:
(-----)    Distribute Subsection #2
           |  Broadcast to Node List
           |  |  Nodes = 53, 122, 156
(-----)    Distribute Subsection #3
           |  Broadcast to Node List
           |  |  Nodes = 53, 122, 156
(-----)    Distribute Subsection #1
           |  Broadcast to Node List
           |  |  Nodes = 53, 122, 156
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
           |  #Columns = 5

        Subsection #1:
(    9)    Access Table Name = QUERY.EMPLOYEE  ID = 5
           |  #Columns = 3
           |  Relation Scan
           |  |  Prefetch: Eligible
           |  Lock Intents
           |  |  Table: Intent Share
           |  |  Row  : Share
(    9)    |  Insert Into Sorted Temp Table  ID = t1
           |  |  #Columns = 3
           |  |  #Sort Key Columns = 1
           |  |  Sortheap Allocation Parameters:
           |  |  |  #Rows     = 1000
           |  |  |  Row Width = 36
           |  |  Piped
(    8)    Sorted Temp Table Completion  ID = t1
(    7)    Access Temp Table  ID = t1
           |  #Columns = 3
           |  Relation Scan
           |  |  Prefetch: Eligible
(    7)    Merge Join
(   12)    |  Access Table Queue  ID = q2  #Columns = 2
           |  |  Output Sorted #Key Columns = 1
```

```
(    5)    Merge Join
(   17)    |  Access Table Queue  ID = q3  #Columns = 3
           |  |  Output Sorted #Key Columns = 1
(    2)    Insert Into Asynchronous Table Queue  ID = q1
           |  Broadcast to Coordinator Node
           |  Rows Can Overflow to Temporary Table

        Subsection #2:
(   14)    Access Table Name = QUERY.PROJECT  ID = 7
           |  #Columns = 2
           |  Relation Scan
           |  |  Prefetch: Eligible
           |  Lock Intents
           |  |  Table: Intent Share
           |  |  Row  : Share
(   14)    |  Insert Into Sorted Temp Table  ID = t2
           |  |  #Columns = 2
           |  |  #Sort Key Columns = 1
           |  |  Sortheap Allocation Parameters:
           |  |  |  #Rows     = 1000
           |  |  |  Row Width = 36
           |  |  Piped
(   13)    Sorted Temp Table Completion  ID = t2
(   12)    Access Temp Table  ID = t2
           |  #Columns = 2
           |  Relation Scan
           |  |  Prefetch: Eligible
(   12)    Insert Into Asynchronous Table Queue  ID = q2
           |  Broadcast to All Nodes of Subsection 1
           |  Rows Can Overflow to Temporary Table

        Subsection #3:
(   19)    Access Table Name = QUERY.DEPARTMENT  ID = 4
           |  #Columns = 3
           |  Relation Scan
           |  |  Prefetch: Eligible
           |  Lock Intents
           |  |  Table: Intent Share
           |  |  Row  : Share
(   19)    |  Insert Into Sorted Temp Table  ID = t3
           |  |  #Columns = 3
           |  |  #Sort Key Columns = 1
           |  |  Sortheap Allocation Parameters:
           |  |  |  #Rows     = 1000
           |  |  |  Row Width = 60
           |  |  Piped
(   18)    Sorted Temp Table Completion  ID = t3
(   17)    Access Temp Table  ID = t3
           |  #Columns = 3
           |  Relation Scan
           |  |  Prefetch: Eligible
(   17)    Insert Into Asynchronous Table Queue  ID = q3
```

```
    |  Broadcast to All Nodes of Subsection 1
    |  Rows Can Overflow to Temporary Table

End of section
```

This plan has all the same pieces as the plan in the first example, but the section has been broken into four subsections. The subsections have the following tasks:

- **Coordinator Subsection**. This subsection coordinates the other subsections. In this plan, it causes the other subsections to be distributed and then uses a table queue to gather the results to be returned to the application.
- **Subsection #1**. This subsection accesses the EMPLOYEE table and builds a piped, sorted temporary table with the results. Just as in the first example, the temporary table is used as the outer table of a merge join. The inner table of the join, however, is not created by this subsection. Instead, the inner table is read from a sorted table queue. The second merge join also reads its inner table from a table queue. After joining all the rows, the results will be sent to the coordinator via a table queue.
- **Subsection #2**. This subsection accesses the PROJECT table and builds a piped, sorted temporary table with the results. The contents of the temporary table are then broadcast to all the nodes that subsection #1 is running on.
- **Subsection #3**. This subsection accesses the DEPARTMENT table and builds a piped, sorted temporary table with the results. The contents of the temporary table are then broadcast to all the nodes that subsection #1 is running on.

It is interesting to examine how the joins are done in the query. In this case, both joins are handled in the same way: the outer table is on local disk and the inner is broadcast to all nodes, which makes these broadcast inner joins. (For more information on join strategies, see “Join Concepts” on page 359.)

Appendix M. National Language Support (NLS)

This appendix contains information about the National Language Support (NLS) provided by DB2, including information about countries, languages, and code pages (code sets) supported and how to configure and use DB2 NLS features in both your applications and databases.

Deriving Code Page Values

The **application code page** is derived from the active environment when the database connection is made. If the DB2CODEPAGE environment variable is set, its value is taken as the application code page. The **database code page** is derived from the value specified (explicitly or by default) at the time the database is created. The following defines how the *active environment* is determined in different operating environments, for example:

UNIX In UNIX-based environments, the active environment is determined from the locale environment variables, which include information about language, territory and code set.

OS/2 In OS/2, primary and secondary code pages are specified in the `CONFIG.SYS` file. You can use the `chcp` command to display and dynamically change code pages within a given session.

DOS In DOS, the active code page is determined by the value specified in the `COUNTRY` command in the `CONFIG.SYS` file. You can use the `chcp` command to display and dynamically change code pages within a given session.

Macintosh For the Macintosh operating system, if the DB2CODEPAGE environment variable is not set, the Macintosh code page is derived from the Regional version code from the installed script.

Windows For Windows, if the DB2CODEPAGE environment variable is not set, the Windows code page is derived from the country ID, as specified in the `iCountry` value in the `[intl]` section of the Windows WIN.INI file.

Windows 95 For Windows 95, if the DB2CODEPAGE environment variable is not set, the Windows 95 code page is derived from the ANSI code page setting in the Registry.

Windows NT For Windows NT, if the DB2CODEPAGE environment variable is not set, the Windows NT code page is derived from the ANSI code page setting in the Registry.

For a complete list of environment mappings for code page values, see the Table 101 on page 815.

Deriving Locales in Application Programs

Locales are specific to UNIX-based operating systems. There are two locales:

- The environment locale allows you to specify the language, currency symbol, and so on, that you want to use.
- The program locale contains the current language, currency symbol, and so on, of a program that is executing.

When your program is started, it gets a default C locale. It does **not** get a copy of the environment locale. Your program has a few choices:

- Ignore the environment locale. Your program could hard code some options. For example, your program could set the language to *spanish* with the `setlocale()` function.
- Copy the environment locale to the program locale.
- Ignore the environment locale. Use whatever defaults you get from the operating system.

How DB2 Derives Locales

With UNIX, the active locale used by DB2 is determined from the LC_CTYPE portion of the locale. For details, see the NLS documentation for your operating system.

- If LC_CTYPE of the program locale has a value other than that of 'C', DB2 will use this value to determine the application code page by mapping it to its corresponding code page.
- If LC_CTYPE has the value of 'C' (the 'C' locale), DB2 will set the program locale according to the environment locale using the `setlocale()` function.
- If LC_CTYPE still has a value of 'C', DB2 will use its default locale for that platform. See either the *Building Applications for Windows and OS/2 Environments* or the *Building Applications for UNIX Environments* book for information on the default locale for that platform.
- If LC_CTYPE's value is no longer 'C', its new value will be used to map to a corresponding code page.

Country Code and Code Page Support

Table 101 on page 815 shows the languages and code sets supported by the Database Servers and how these values are mapped to country code and code page values that are used by the database manager.

The following is an explanation of each column of the table:

1. **Code Page** shows the IBM-defined code page as mapped from the operating system code set.
2. **Code Set** shows the code set associated with the supported language. The code set is mapped to the DB2 Code Page.

3. **Country Code** shows the country code that is used by the database manager internally for providing country-specific support.
4. **Locale** shows the locale values supported by the database manager.
5. **Operating System** shows the operating system that supports the languages and code sets.

Table 101 (Page 1 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
ARABIC COUNTRIES - AA				
864	IBM-864	785	-	OS/2
1046	IBM-1046	785	Ar_AA	AIX
1089	ISO8859-6	785	ar_AA	AIX
1089	iso88596	785	ar_SA.iso88596	HP
1256	1256	785	-	WIN
420	IBM-420	785	-	HOST
AUSTRALIA - AU				
437	IBM-437	61	-	OS/2
850	IBM-850	61	-	OS/2
819	ISO8859-1	61	en_AU	AIX
850	IBM-850	61	En_AU	AIX
819	iso8859–1	61	-	HP
1051	roman8	61	-	HP
819	ISO8859-1	61	-	Sun
1252	1252	61	-	WIN
1275	1275	61	-	Mac
37	IBM037	61	-	HOST
AUSTRIA - AT				
437	IBM-437	43	-	OS/2
850	IBM-850	43	-	OS/2
819	ISO8859-1	43	ge_AT	AIX
850	IBM-850	43	Ge_AT	AIX
819	iso88591	43	-	HP
1051	roman8	43	-	HP
819	ISO8859-1	43	-	Sun
1252	1252	43	-	WIN
1275	1275	43	-	Mac
37	IBM-037	43	-	HOST
BELGIUM - BE				

Table 101 (Page 2 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
437	IBM-437	32	-	OS/2
850	IBM-850	32	-	OS/2
819	ISO8859-1	32	nl_BE fr_BE	AIX
850	IBM-850	32	Nl_BE Fr_BE	AIX
819	iso88591	32	-	HP
819	ISO8859-1	32	-	Sun
1252	1252	32	-	WIN
1275	1275	32	-	Mac
500	IBM-500	32	-	HOST
BULGARIA - BG				
855	IBM-855	359	-	OS/2
915	ISO8859-5	359	-	OS/2
915	ISO8859-5	359	bg_BG	AIX
915	iso88595	359	-	HP
1251	1251	359	-	WIN
1283	1283	359	-	Mac
1025	IBM-1025	359	-	HOST
BRAZIL - BR				
850	IBM-850	55	-	OS/2
850	IBM-850	55	Pt_BR	AIX
819	ISO8859-1	55	pt_BR	AIX
819	ISO8859-1	55	-	HP
819	ISO8859-1	55	-	Sun
1252	1252	55	-	WIN
37	IBM-037	55	-	HOST
CANADA - CA				
850	IBM-850	1	-	OS/2
850	IBM-850	1	En_CA	AIX
819	ISO8859-1	1	en_CA	AIX
819	iso88591	1	fr_CA.iso88591	HP
1051	roman8	1	fr_CA.roman8	HP
819	ISO8859-1	1	-	Sun
1252	1252	1	-	WIN
1275	1275	1	-	Mac
37	IBM-037	1	-	HOST

Table 101 (Page 3 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
863	IBM-863	2	-	OS/2 (French)
CHINA (PRC) - CN				
1381	IBM-1381	86	-	OS/2
1386	GBK	86	-	OS/2
1383	IBM-eucCN	86	zh_CN	AIX
1386	GBK	86	Zh_CN.GBK	AIX
1383	IBM-eucCN	86	zh_CN.hp15CN	HP
1383	euc-CN	86	zh	Sun
1381	IBM-1381	86	-	WIN
1386	GBK	86	-	WIN
935	IBM-935	86	-	HOST
CROATIA - HR				
852	IBM-852	385	-	OS/2
912	ISO8859-2	385	hr_HR	AIX
912	iso88592	385	hr_HR.iso88592	HP
1250	1250	385	-	WIN
1282	1282	385	-	Mac
870	IBM-870	385	-	HOST
CZECH REPUBLIC - CZ				
852	IBM-852	42	-	OS/2
912	ISO8859-2	42	cs_CZ	AIX
912	iso88592	42	cs_CZ.iso88592	HP
1250	1250	42	-	WIN
1282	1282	42	-	Mac
870	IBM-870	42	-	HOST
DENMARK - DK				
850	IBM-850	45	-	OS/2
819	ISO8859-1	45	da_DK	AIX
850	IBM-850	45	Da_DK	AIX
819	iso88591	45	da_DK.iso88591	HP
1051	roman8	45	da_DK.roman8	HP
819	ISO8859-1	45	-	Sun
1252	1252	45	-	WIN
1275	1275	45	-	Mac
277	IBM-277	45	-	HOST

Table 101 (Page 4 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
ESTONIA - EE				
922	IBM-922	372	-	OS/2
922	IBM-922	372	Et_EE	AIX
922	IBM-922	372	-	WIN
FINLAND - FI				
437	IBM-437	358	-	OS/2
850	IBM-850	358	-	OS/2
819	ISO8859-1	358	fi_FI	AIX
850	IBM-850	358	Fi_FI	AIX
819	iso88591	358	fi_FI.iso88591	HP
819	ISO8859-1	358	-	Sun
1051	roman8	358	-	HP
1252	1252	358	-	WIN
1275	1275	358	-	Mac
278	IBM-278	358	-	HOST
FRANCE - FR				
437	IBM-437	33	-	OS/2
850	IBM-850	33	-	OS/2
819	ISO8859-1	33	fr_FR	AIX
850	IBM-850	33	Fr_FR	AIX
819	iso88591	33	fr_FR.iso88591	HP
1051	roman8	33	fr_FR.roman8	HP
819	ISO8859-1	33	fr	Sun
819	ISO8859-1	33	fr_FR.ISO8859-1	SCO
1252	1252	33	-	WIN
1275	1275	33	-	Mac
297	IBM-297	33	-	HOST
GERMANY - DE				
437	IBM-437	49	-	OS/2
850	IBM-850	49	-	OS/2
819	ISO8859-1	49	de_DE	AIX
850	IBM-850	49	De_DE	AIX
819	iso88591	49	de_DE.iso88591	HP
1051	roman8	49	de_De.roman8	HP
819	ISO8859-1	49	de	Sun

Table 101 (Page 5 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
819	ISO8859-1	49	de_DE.ISO8859-1	SCO
1252	1252	49	-	WIN
1275	1275	49	-	Mac
273	IBM-273	49	-	HOST
819	ISO8859-1	49	De_DE.88591	SINIX
819	ISO8859-1	49	De_DE.6937	SINIX
GREECE - GR				
813	ISO8859-7	30	-	OS/2
869	IBM-869	30	-	OS/2
813	ISO8859-7	30	el_GR	AIX
813	iso8859-7	30	el_GR.iso88597	HP
1253	1253	30	-	WIN
1280	1280	30	-	Mac
423	IBM-423	30	-	HOST
875	IBM-875	30	-	HOST
HUNGARY - HU				
852	IBM-852	36	-	OS/2
912	ISO8859-2	36	hu_HU	AIX
912	iso88592	36	hu_HU.iso88592	HP
1250	1250	36	-	WIN
1282	1282	36	-	Mac
870	IBM-870	36	-	HOST
ICELAND - IS				
850	IBM-850	354	-	OS/2
819	ISO8859-1	354	is_IS	AIX
850	IBM-850	354	Is_IS	AIX
819	iso88591	354	is_IS.iso88591	HP
1051	roman8	354	is_IS.roman8	HP
819	ISO8859-1	354	-	Sun
1252	1252	354	-	WIN
1275	1275	354	-	Mac
871	IBM-871	354	-	HOST
IRELAND - IE				
437	IBM-437	353	-	OS/2
850	IBM-850	353	-	OS/2

Table 101 (Page 6 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
819	ISO8859-1	353	en_IE	AIX
850	IBM-850	353	En_IE	AIX
819	iso88591	353	-	HP
1051	roman8	353	-	HP
819	ISO8859-1	353	-	Sun
819	ISO8859-1	353	en_IE.ISO8859-1	SCO
1252	1252	353	-	WIN
1275	1275	353	-	Mac
285	IBM-285	353	-	HOST
ISRAEL - IL				
862	IBM-862	972	-	OS/2
916	ISO8859-8	972	iw_IL	AIX
1255	1255	972	-	WIN
424	IBM-424	972	-	HOST
ITALY - IT				
437	IBM-437	39	-	OS/2
850	IBM-850	39	-	OS/2
819	ISO8859-1	39	It_IT	AIX
850	IBM-850	39	It_IT	AIX
819	iso88591	39	it_IT.iso88591	HP
1051	roman8	39	it_IT.roman8	HP
819	ISO8859-1	39	it	Sun
1252	1252	39	-	WIN
1275	1275	39	-	Mac
280	IBM-280	39	-	HOST
JAPAN - JP				
932	IBM-932	81	-	OS/2
942	IBM-942	81	-	OS/2
943	IBM-943	81	-	OS/2
954	IBM-eucJP	81	ja_JP	AIX
932	IBM-932	81	Ja_JP	AIX
954	eucJP	81	ja_JP.eucJP	HP
5039	SJIS	81	ja_JP.SJIS	HP
954	eucJP	81	ja	Sun
943	IBM-943	81	-	WIN

Table 101 (Page 7 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
930	IBM-930	81	-	HOST
939	IBM-939	81	-	HOST
5026	IBM-5026	81	-	HOST
5035	IBM-5035	81	-	HOST
LATIN AMERICA - LAT				
437	IBM-437	3	-	OS/2
850	IBM-850	3	-	OS/2
819	ISO8859-1	3	-	AIX
850	IBM-850	3	-	AIX
819	iso88591	3	-	HP
819	ISO8859-1	3	-	Sun
1051	roman8	3	-	HP
1252	1252	3	-	WIN
1275	1275	3	-	Mac
284	IBM-284	3	-	HOST
LATVIA - LV				
921	IBM-921	371	-	OS/2
921	IBM-921	371	Lv_LV	AIX
921	IBM-921	371	-	WIN
LITHUANIA - LT				
921	IBM-921	370	-	OS/2
921	IBM-921	370	Lt_LT	AIX
921	IBM-921	370	-	WIN
MACEDONIA (FYR) - MK				
855	IBM-855	389	-	OS/2
915	ISO8859-5	389	-	OS/2
915	ISO8859-5	389	mk_MK	AIX
915	iso88595	389	-	HP
1251	1251	389	-	WIN
1283	1283	389	-	Mac
1025	IBM-1025	389	-	HOST
NETHERLANDS - NL				
437	IBM-437	31	-	OS/2
850	IBM-850	31	-	OS/2
819	ISO8859-1	31	nl_NL	AIX

Table 101 (Page 8 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
850	IBM-850	31	Nl_NL	AIX
819	iso8859-1	31	nl_NL.iso88591	HP
1051	roman8	31	n1_NL.roman8	HP
819	ISO8859-1	31	nl	Sun
1252	1252	31	-	WIN
1275	1275	31	-	Mac
37	IBM-037	31	-	HOST
NEW ZEALAND - NZ				
850	IBM-850	64	-	OS/2
850	IBM-850	64	En_NZ	AIX
819	ISO8859-1	64	en_NZ	AIX
819	ISO8859-1	64	-	HP
819	ISO8859-1	64	-	Sun
1252	1252	64	-	WIN
37	IBM-037	64	-	HOST
NORWAY - NO				
850	IBM-850	47	-	OS/2
819	ISO8859-1	47	no_NO	AIX
850	IBM-850	47	No_NO	AIX
819	iso88591	47	no_NO.iso88591	HP
1051	roman8	47	no_NO.roman8	HP
819	ISO8859-1	47	-	Sun
1252	1252	47	-	WIN
1275	1275	47	-	Mac
277	IBM-277	47	-	HOST
POLAND - PL				
852	IBM-852	48	-	OS/2
912	ISO8859-2	48	pl_PL	AIX
912	iso8859-2	48	pl_PL.iso88592	HP
1250	1250	48	-	WIN
1282	1282	48	-	Mac
870	IBM-870	48	-	HOST
PORTUGAL - PT				
860	IBM-860	351	-	OS/2
850	IBM-850	351	-	OS/2

Table 101 (Page 9 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
819	ISO8859-1	351	pt_PT	AIX
850	IBM-850	351	Pt_PT	AIX
819	iso8859-1	351	pt_PT.iso88591	HP
1051	roman8	351	pt_PT.roman8	HP
819	ISO8859-1	351	-	Sun
1252	1252	351	-	WIN
1275	1275	351	-	Mac
37	IBM-037	351	-	HOST
ROMANIA - RO				
852	IBM-852	40	-	OS/2
912	ISO8859-2	40	ro_RO	AIX
912	iso8859-2	40	ro_RO.iso88592	HP
1250	1250	40	-	WIN
1282	1282	40	-	Mac
870	IBM-870	40	-	HOST
RUSSIA - RU				
866	IBM-866	7	-	OS/2
915	ISO8859-5	7	-	OS/2
915	ISO8859-5	7	ru_RU	AIX
915	iso8859-5	7	ru_RU.iso88585	HP
1251	1252	7	-	WIN
1283	1283	7	-	Mac
1025	IBM-1025	7	-	HOST
SERBIA/MONTENEGRO - SP				
855	IBM-855	381	-	OS/2
915	ISO8859-5	381	-	OS/2
915	ISO8859-5	381	sr_SP	AIX
915	iso8859-5	381	-	HP
1251	1251	381	-	WIN
1283	1283	381	-	Mac
1025	IBM-1025	381	-	HOST
SLOVAKIA - SK				
852	IBM-852	938	-	OS/2
912	ISO8859-2	938	sk_SK	AIX
912	iso8859-2	938	sk_SK.iso88592	HP

Table 101 (Page 10 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
1250	1250	938	-	WIN
1282	1282	938	-	Mac
870	IBM-870	938	-	HOST
SLOVENIA - SI				
852	IBM-852	386	-	OS/2
912	ISO8859-2	386	sl_SL	AIX
912	iso8859-2	386	sl_SL.iso88592	HP
1250	1250	386	-	WIN
1282	1282	386	-	Mac
870	IBM-870	386	-	HOST
SOUTH AFRICA - ZA				
437	IBM-437	227	-	OS/2
850	IBM-850	27	-	OS/2
819	ISO8859-1	27	en_ZA	AIX
850	IBM-850	27	En_ZA	AIX
819	iso8859-1	27	-	HP
1051	roman8	27	-	HP
819	ISO8859-1	27	-	Sun
819	ISO8859-1	27	en_ZA.ISO8859-1	SCO
1252	1252	27	-	WIN
1275	1275	27	-	Mac
285	IBM-285	27	-	HOST
SOUTH KOREA - KR				
949	IBM-949	82	-	OS/2
970	IBM-eucKR	82	ko_KR	AIX
970	eucKR	82	ko_KR.eucKR	HP
970	eucKR	82	ko	Sun
1363	1363	82	-	WIN
933	IBM-933	82	-	HOST
SPAIN - ES				
437	IBM-437	34	-	OS/2
850	IBM-850	34	-	OS/2
819	ISO8859-1	34	es_ES	AIX
850	IBM-850	34	Es_ES	AIX
819	iso8859-1	34	es_ES.iso88591	HP

Table 101 (Page 11 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
1051	roman8	34	es_ES.roman8	HP
819	ISO8859-1	34	es	Sun
819	ISO8859-1	34	es_ES.ISO8859-1	SCO
1252	1252	34	-	WIN
1275	1275	34	-	Mac
284	IBM-284	34	-	HOST
SWEDEN - SE				
437	IBM-437	46	-	OS/2
850	IBM-850	46	-	OS/2
819	ISO8859-1	46	sv_SE	AIX
850	IBM-850	46	Sv_SE	AIX
819	iso88591	46	sv_SE.iso88591	HP
1051	roman8	46	sv_SE.roman8	HP
819	ISO8859-1	46	sv	Sun
1252	1252	46	-	WIN
1275	1275	46	-	Mac
278	IBM-278	46	-	HOST
SWITZERLAND - CH				
437	IBM-437	41	-	OS/2
850	IBM-850	41	-	OS/2
819	ISO8859-1	41	de_CH	AIX
850	IBM-850	41	De_CH	AIX
819	iso88591	41	-	HP
1051	roman8	41	-	HP
819	ISO8859-1	41	-	Sun
1252	1252	41	-	WIN
1275	1275	41	-	Mac
500	IBM-500	41	-	HOST
TAIWAN - TW				
938	IBM-938	886	-	OS/2
948	IBM-948	886	-	OS/2
950	big5	886	-	OS/2
950	big5	886	Zh_TW	AIX
964	IBM-eucTW	886	Zh_TW	AIX
950	big5	886	zh_TW.big5	HP

Table 101 (Page 12 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
964	eucTW	886	zh_TW.eucTW	HP
950	big5	886	big5	Sun
964	eucTW	886	zh_TW	Sun
950	big5	886	-	Win
937	IBM-937	886	-	HOST
THAILAND - TH				
874	TIS620-1	66	-	OS/2
874	TIS620-1	66	Th_TH	AIX
874	tis620	66	th_TH.tis620	HP
874	TIS620-1	66	-	WIN
838	IBM-838	66	-	HOST
TURKEY - TR				
857	IBM-857	90	-	OS/2
920	ISO8859-9	90	tr_TR	AIX
920	ISO8859-9	90	tr_TR.iso88599	HP
1254	1254	90	-	WIN
1281	1281	90	-	Mac
1026	IBM-1026	90	-	HOST
UNITED KINGDOM - GB				
437	IBM-437	44	-	OS/2
850	IBM-850	44	-	OS/2
819	ISO8859-1	44	en_GB	AIX
850	IBM-850	44	En_GB	AIX
819	iso88591	44	en_GB.iso88591	HP
1051	roman8	44	en_GB.roman8	HP
819	ISO8859-1	44	-	Sun
819	ISO8859-1	44	en_GB.ISO8859-1	SCO
819	88591	44	En_GB.88591	SINIX
819	ISO8859-1	44	En_GB.6937	SINIX
1252	1252	44	-	WIN
1275	1275	44	-	Mac
285	IBM-285	44	-	HOST
United States of America - US				
437	IBM-437	1	-	OS/2
850	IBM-850	1	-	OS/2

Table 101 (Page 13 of 13). Supported Languages and Code Sets

Code Page	Code Set	Ctry. Code	Locale	Op. System
819	ISO8859-1	1	en_US	AIX
850	IBM-850	1	En_US	AIX
819	iso8859-1	1	en_US.iso88591	HP
1051	roman8	1	en_US.roman8	HP
819	ISO8859-1	1	en_US	Sun
819	ISO8859-1	1	en_US	SGI
819	ISO8859-1	1	en_US.ISO8859-1	SCO
1252	1252	1	-	WIN
1275	1275	1	-	Mac
37	IBM-037	1	-	HOST

Note: The Solaris code page 950 does not support the following characters in IBM 950:

Code Range	Description	Sun Big-5	IBM Big-5
C6A1-C8FE	Symbols	Reserved area	Symbols
F9D6-F9FE	ETen extension	Reserved area	ETen extension
F286-F9A0	IBM selected chars	Reserved area	IBM selected

Character Sets

The database manager does not, in general, restrict the character set available to an application except as noted below.

DBCS Character Sets

Each combined SBCS/DBCS code page allows for both single- and double-byte character code points. This is accomplished by reserving a subset of the 256 available code points of each implied SBCS code page identifier for single-byte characters, with the remainder of the code points either undefined or allocated to the first byte of double-byte code points. These code points are shown in the following table.

Table 102. Mixed Character Set Code Points		
Supported Mixed Code Page	**Code Points for Single-byte Characters**	**Code Points for First Byte of Double-Byte Characters**
932	x00-7F, xA1-DF	x81-9F, xE0-FC
942, 943	x00-80, xA0-DF, xFD-FF	x81-9F, xE0-FC
938	x00-7E	x81-FC
948	x00-80	x81-FC
949	x00-7F	x8F-FE
950	x00-7E	x81-FE
1381	x00-7F	x8C-FE

Code points not assigned to either category above are not defined, and are processed as single-byte undefined code points.

Within each implied DBCS code page, there are 256 code points available as the second byte for each valid first byte. These code points are also partitioned into valid and invalid second byte ranges for the purpose of determining whether a DBCS character is properly formed. Note that in DBCS environments, DB2 does not perform validity checking on individual double-byte characters.

Character Set for Identifiers

The basic character set that may be used in database names consists of the single-byte uppercase and lowercase Latin letters (A...Z, a...z), the Arabic numerals (0...9) and the underscore character (_). This list of letters is augmented with the three special characters #, @ and $ to provide compatibility with host database products. However, these special characters should be used with care in an NLS environment because they are not included in the NLS host (EBCDIC) invariant character set.

When naming database objects (such as tables and views), program labels, host variables, cursors and statements alphabetics from the extended character set may also be used. For example, those letters with diacritical marks. The available characters depend on the code page in use and if you are using the database in a multiple code page environment, you must ensure that all code pages support any alphabetics you plan on using from the extended character set. See the *SQL Reference* for a discussion of delimited identifiers which can be used in SQL statements and can contain characters outside the extended character set.

Extended Character Set Definition for DBCS Identifiers

In DBCS environments, the extended character set consists of all the characters in the basic character set, plus those identified as a letter or digit as follows:

- All double-byte characters in each DBCS code page, except the double-byte space, are valid letters.
- The double-byte space is a special character.

- The single-byte characters available in each mixed code page are assigned to various categories as follows:

Category	Valid Code Points within each Mixed Code Page
Digits	x30-39
Letters	x23-24, x40-5A, x61-7A, xA6-DF (A6-DF for code pages 932 and 942 only)
Special Characters	All other valid single-byte character code points

Coding of SQL Statements

The coding of SQL statements is not language dependent. SQL is a programming language and, like other programming languages such as C, it is language invariant. The SQL keywords must be typed as shown, although they may be typed in uppercase, lowercase, or mixed case. The names of database objects, host variables and program labels that occur in an SQL statement cannot contain characters outside the database manager extended character set as described above.

Collating Sequences

The database manager compares character data using a *collating sequence*. This is an ordering for a set of characters that determines whether a particular character sorts higher, lower, or the same as another.

Note: Character string data defined with the FOR BIT DATA attribute, or BLOB data, is sorted using the binary sort sequence.

For example, a collating sequence can be used to indicate that lowercase and uppercase versions of a particular character are to be sorted equally.

The database manager allows databases to be created with custom collating sequences. The following sections help you determine and implement a particular collating sequence for a database.

Overview

In a database, each single-byte character is represented internally as a unique number between 0 and 255, (in hexadecimal notation, between X'00' and X'FF'). This number is referred to as the *code point* of the character. A collating sequence is a mapping between the code point and the desired position of each character in a sorted sequence. The numeric value of the position is called the *weight* of the character in the collating sequence. The simplest collating sequence is one where the weights are identical to the code points. This is called the *identity sequence*.

For example, consider the characters B (X'42'), and b (X'62'). If, according to the collating sequence table, they both have a sort weight of X'42' (B), then they collate the same. If the sort weight for B is X'9E' and the sort weight for b is X'9D', then b will be sorted before B. Actual weights depend on the collating sequence table used which depends on the code set and locale. Note that a collating sequence table is not the same as a code page table which defines code points.

Consider the following example. In ASCII, the characters A through Z are represented by X'41' through X'5A'. To describe a collating sequence where these are sorted in order, and consecutively (no intervening characters), you can write X'41', X'42', ...X'59', X'5A'.

For multi-byte characters, the hexadecimal value of the multi-byte character is also used as the weight. For example, X'8260', X'8261' are the code points for double byte character A and B. In this case, you can write X'8260', X'8261' as the collating sequence for double byte characters A and B. These are also the code points for A and B.

The values of the weights in a collating sequence need not be unique. For example, you could give uppercase letters and their lowercase equivalents the same weight.

Specifying the collating sequence can be simplified if a collating sequence provides weights for all 256 code points. The weight of each character can be determined using the code point of the character. This is the method used to specify a collating sequence for the database manager: a string of 256 bytes, where the *nth* byte (starting with 0) contains the weight of code point *n*.

In the case of multi-byte character sets, DB2 simply uses the code point as the collating value. Multi-byte characters therefore sort the way they appear in their code point table.

Character Comparisons

Once a collating sequence is established, character comparison is performed by comparing the weights of two characters, instead of directly comparing their code point values.

If weights that are not unique are used, characters that are not identical may compare equally. Because of this, string comparison must be a two-phase process:

1. Compare the characters of each string based on their weights.
2. If step 1 yielded equality, compare the characters of each string based on their code point values.

If the collating sequence contains 256 unique weights, only the first step is performed. If the collating sequence is the identity sequence only the second step is performed. In either case, there is a performance benefit.

For more information on character comparisons, see the *SQL Reference*.

Case Independent Comparisons

To perform character comparisons that are independent of whether they are upper or lower case, you can use the TRANSLATE function to select and compare mixed case column data by translating it to upper case, but only for the purposes of comparison. Consider the following data:

```
Abel
abels
ABEL
abel
ab
Ab
```

For the following select statement:

```
SELECT c1 FROM T1 WHERE TRANSLATE(c1) LIKE 'AB%'
```

you would receive the following results:

```
ab
Ab
abel
Abel
ABEL
abels
```

Note: You could also set the select as in the following view v1, and then make all your comparisons against the view (in upper case) and your inserts into the table in mixed case:

```
CREATE VIEW v1 AS SELECT TRANSLATE(c1) FROM t1
```

At the database level, you can set the *collating sequence* as part of the CREATE DATABASE API . This allows you to decide if 'a' is processed before 'A', or if 'A' is processed after 'a', or if they are processed with equal weighting. This will make them equal when collating or sorting using the ORDER BY clause. If you have two values of 'a' and 'A', 'A' will always come before 'a', because in all senses they are equal, so the only difference upon which to sort is the hexadecimal value.

Thus if you issue SELECT c1 FROM t1 WHERE c1 LIKE 'ab%', you receive the following output:

```
ab
abel
abels
```

If you issue SELECT c1 FROM t1 WHERE c1 LIKE 'A%', you receive the following output:

```
Abel
Ab
ABEL
```

If you issue SELECT c1 FROM t1 ORDER BY c1, you receive the following:

```
ab
Ab
abel
Abel
ABEL
abels
```

Thus, you may want to consider using the scalar function TRANSLATE(), as well as the CREATE DATABASE API. Note that you can only specify a collating sequence using the CREATE DATABASE API. You cannot specify a collating sequence from the Command Line Processor. For information on the TRANSLATE() function, see the *SQL Reference*. For information on the CREATE DATABASE API see the *API Reference*.

You can also use the UCASE function as follows, but note that DB2 performs a table scan instead of using an index for the select:

```
SELECT * FROM EMP WHERE UCASE(JOB) = 'NURSE'
```

Specifying a Collating Sequence

The collating sequence for a database is specified at database creation time. Once the database has been created, the collating sequence cannot be changed.

The CREATE DATABASE API accepts a data structure called the Database Descriptor Block (SQLEDBDESC). You can define your own collating sequence within this structure.

To specify a collating sequence for a database:

- Pass the desired SQLEDBDESC structure, or
- Pass a NULL pointer. The collating sequence of the operating system (based on current country code and code page) is used. This is the same as specifying SQLDBCSS equal to SQL_CS_SYSTEM (0).

The SQLEDBDESC structure contains:

SQLDBCSS A 4-byte integer indicating the source of the database collating sequence. Valid values are:

SQL_CS_SYSTEM The collating sequence of the operating system (based on current country code and code page) is used.

SQL_CS_USER The collating sequence is specified by the value in the SQLDBUDC field.

SQL_CS_NONE The collating sequence is the identity sequence. Strings are compared byte for byte, starting with the first byte, using a simple binary comparison.

Note: These constants are defined in the SQLENV include file.

SQLDBUDC A 256-byte field. The nth byte contains the sort weight of the nth character in the code page of the database. If SQLDBCSS is not equal to SQL_CS_USER, this field is ignored.

Sample Collating Sequences

Several sample collating sequences are provided (as include files) to facilitate database creation using the EBCDIC collating sequences instead of the default workstation collating sequence.

The collating sequences in these include files can be specified in the SQLDBUDC field of the SQLEDBDESC structure. They can also be used as models for the construction of other collating sequences.

Other Concerns

Once a collating sequence is defined, all future character comparisons for that database will be performed with that collating sequence. Except for character data defined as FOR BIT DATA or BLOB data, the collating sequence will be used for all SQL comparisons and ORDER BY clauses, and also in setting up indexes and statistics. For more information on how the database collating sequence is used, see the section on *String Comparisons* in the *SQL Reference*, S10J-8165-00.

Potential problems may occur in the following cases:

- An application merges sorted data from a database with application data that was sorted using a different collating sequence.
- An application merges sorted data from one database with sorted data from another, but the databases have different collating sequences.
- An application makes assumptions about sorted data that are not true for the relevant collating sequence. For example, numbers collating lower than alphabetics might or might not be true for a particular collating sequence.

A final point to remember is that the results of any sort based on a direct comparison of characters will only match the results of a query ordered using an identity collating sequence.

Datetime Values

The datetime data types are described below. Although datetime values can be used in certain arithmetic and string operations and are compatible with certain strings, they are neither strings nor numbers.

Date

A *date* is a three-part value (year, month, and day). The range of the year part is 0001 to 9999. The range of the month part is 1 to 12. The range of the day part is 1 to *x*, where *x* depends on the month.

The internal representation of a date is a string of 4 bytes. Each byte consists of 2 packed decimal digits. The first 2 bytes represent the year, the third byte the month, and the last byte the day.

The length of a DATE column, as described in the SQLDA, is 10 bytes, which is the appropriate length for a character string representation of the value.

Time

A *time* is a three-part value (hour, minute, and second) designating a time of day under a 24-hour clock. The range of the hour part is 0 to 24; while the range of the other parts is 0 to 59. If the hour is 24, the minute and second specifications will be zero.

The internal representation of a time is a string of 3 bytes. Each byte is 2 packed decimal digits. The first byte represents the hour, the second byte the minute, and the last byte the second.

The length of a TIME column, as described in the SQLDA, is 8 bytes, which is the appropriate length for a character string representation of the value.

Timestamp

A *timestamp* is a seven-part value (year, month, day, hour, minute, second, and microsecond) that designates a datetime as defined above, except that the time includes a fractional specification of microseconds.

The internal representation of a timestamp is a string of 10 bytes, each of which consists of 2 packed decimal digits. The first 4 bytes represent the date, the next 3 bytes the time, and the last 3 bytes the microseconds.

The length of a TIMESTAMP column, as described in the SQLDA, is 26 bytes, which is the appropriate length for the character string representation of the value.

String Representations of Datetime Values

Values whose data types are DATE, TIME, or TIMESTAMP are represented in an internal form that is transparent to the SQL user. Dates, times, and timestamps can, however, also be represented by character strings, and these representations directly concern the SQL user since there are no constants or variables whose data types are DATE, TIME, or TIMESTAMP. Thus, to be retrieved, a datetime value must be assigned to a character string variable. The character string representation is normally the default format of datetime values associated with the country code of the database, unless overridden by specification of the *F* format option when the program is precompiled or bound to the database. See Table 105 on page 837 for a listing of the string formats for the various country codes.

When a valid string representation of a datetime value is used in an operation with an internal datetime value, the string representation is converted to the internal form of the date, time, or timestamp before the operation is performed. The following sections define the valid string representations of datetime values.

Date Strings

A string representation of a date is a character string that starts with a digit and has a length of at least 8 characters. Trailing blanks may be included; leading zeros may be omitted from the month and day portions.

Valid string formats for dates are listed in Table 1. Each format is identified by name and includes an associated abbreviation and an example of its use.

Table 103. Formats for String Representations of Dates			
Format Name	**Abbreviation**	**Date Format**	**Example**
International Standards Organization	ISO	yyyy-mm-dd	1991-10-27
IBM USA standard	USA	mm/dd/yyyy	10/27/1991
IBM European standard	EUR	dd.mm.yyyy	27.10.1991
Japanese Industrial Standard Christian era	JIS	yyyy-mm-dd	1991-10-27
Site-defined (Local)	LOC	Depends on database country code	—

Time Strings

A string representation of a time is a character string that starts with a digit and has a length of at least 4 characters. Trailing blanks may be included; a leading zero may be omitted from the hour part of the time and seconds may be omitted entirely. If you choose to omit seconds, an implicit specification of 0 seconds is assumed. Thus, 13.30 is equivalent to 13.30.00.

Valid string formats for times are listed in Table 104. Each format is identified by name and includes an associated abbreviation and an example of its use.

Table 104. Formats for String Representations of Times			
Format Name	**Abbreviation**	**Time Format**	**Example**
International Standards Organization	ISO	hh.mm.ss	13.30.05
IBM USA standard	USA	hh:mm AM or PM	1:30 PM
IBM European standard	EUR	hh.mm.ss	13.30.05
Japanese Industrial Standard Christian Era	JIS	hh:mm:ss	13:30:05
Site-defined (Local)	LOC	Depends on application country code	—

Notes:

1. In ISO, EUR and JIS format, .ss (or :ss) is optional.
2. In the case of the USA time string format, the minutes specification may be omitted, indicating an implicit specification of 00 minutes. Thus 1 PM is equivalent to 1:00 PM.
3. In the USA time format, the hour must not be greater than 12 and cannot be 0 except for the special case of 00:00 AM. Using the ISO format of the 24-hour clock, the correspondence between the USA format and the 24-hour clock is as follows:

12:01 AM through 12:59 AM corresponds to 00.01.00 through 00.59.00.

01:00 AM through 11:59 AM corresponds to 01.00.00 through 11.59.00.

12:00 PM (noon) through 11:59 PM corresponds to 12.00.00 through 23.59.00.

12:00 AM (midnight) corresponds to 24.00.00 and 00:00 AM (midnight) corresponds to 00.00.00.

Timestamp Strings

A string representation of a timestamp is a character string that starts with a digit and has a length of at least 16 characters. The complete string representation of a timestamp has the form *yyyy-mm-dd-hh.mm.ss.nnnnnn*. Trailing blanks may be included. Leading zeros may be omitted from the month, day, and hour part of the timestamp, and microseconds may be truncated or entirely omitted. If you choose to omit any digit of the microseconds portion, an implicit specification of 0 is assumed. Thus, 1991-3-2-8.30.00 is equivalent to 1991-03-02-08.30.00.000000.

MBCS Considerations

Date and timestamp strings must contain only single-byte characters and digits.

Date and Time Formats

The character string representation of date and time formats is the default format of datetime values associated with the country code of the application. This default format may be overridden by specification of the *F* format option when the program is precompiled or bound to the database.

The following is a description of the input and output formats for date and time:

- Input Time Format
  - There is no default input time format
  - All time formats are allowed as input for all country codes.
- Output Time Format
  - The default output time format is equal to the local time format.
- Input Date Format
  - There is no default input date format
  - Where the local format for date conflicts with an ISO, JIS, EUR, or USA date format, the local format is recognized for date input. For example, see the UK entry in Table 105 on page 837.
- Output Date Format
  - The default output date format is shown in Table 105 on page 837.

    Note: Table 105 on page 837 also shows a listing of the string formats for the various country codes.

Table 105 (Page 1 of 2). Date and Time Formats by Country Code

Country Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
785 Arabic	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
001 Australia (1)	mm-dd-yyyy	JIS	LOC	LOC, USA, EUR, ISO
061 Australia	dd-mm-yyyy	JIS	LOC	LOC, USA, EUR, ISO
032 Belgium	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
055 Brazil	dd.mm.yy	JIS	LOC	LOC, USA, EUR, ISO
359 Bulgaria	dd.mm.yyyy	JIS	EUR	LOC, USA, EUR, ISO
001 Canada	mm-dd-yyyy	JIS	USA	LOC, USA, EUR, ISO
002 Canada (French)	dd-mm-yyyy	ISO	ISO	LOC, USA, EUR, ISO
385 Croatia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
042 Czech Republic	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
045 Denmark	dd-mm-yyyy	ISO	ISO	LOC, USA, EUR, ISO
358 Finland	dd/mm/yyyy	ISO	EUR	LOC, EUR, ISO
389 FYR Macedonia	dd.mm.yyyy	JIS	EUR	LOC, USA, EUR, ISO
033 France	dd/mm/yyyy	JIS	EUR	LOC, EUR, ISO
049 Germany	dd/mm/yyyy	ISO	ISO	LOC, EUR, ISO
030 Greece	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
036 Hungary	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
354 Iceland	dd-mm-yyyy	JIS	LOC	LOC, USA, EUR, ISO
972 Israel	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
039 Italy	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
081 Japan	mm/dd/yyyy	JIS	ISO	LOC, USA, EUR, ISO
082 Korea	mm/dd/yyyy	JIS	ISO	LOC, USA, EUR, ISO

Table 105 (Page 2 of 2). Date and Time Formats by Country Code

Country Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
001 Latin America (1)	mm-dd-yyyy	JIS	LOC	LOC, USA, EUR, ISO
003 Latin America	dd-mm-yyyy	JIS	LOC	LOC, EUR, ISO
031 Netherlands	dd-mm-yyyy	JIS	LOC	LOC, USA, EUR, ISO
047 Norway	dd/mm/yyyy	ISO	EUR	LOC, EUR, ISO
048 Poland	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
351 Portugal	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
086 PRC	mm/dd/yyyy	JIS	ISO	LOC, USA, EUR, ISO
040 Romania	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
007 Russia	dd/mm/yyyy	ISO	LOC	LOC, EUR, ISO
381 Serbia/Montenegro	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
042 Slovakia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
386 Slovenia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
034 Spain	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
046 Sweden	dd/mm/yyyy	ISO	ISO	LOC, EUR, ISO
041 Switzerland	dd/mm/yyyy	ISO	EUR	LOC, EUR, ISO
088 Taiwan	mm-dd-yyyy	JIS	ISO	LOC, USA, EUR, ISO
066 Thailand (2)	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
090 Turkey	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
044 UK	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
001 USA	mm-dd-yyyy	JIS	USA	LOC, USA, EUR, ISO

Notes:

1. Countries using the default C locale are assigned country code 001.
2. yyyy is in Buddhist era: Gregorian + 543 years.

Appendix N. Splitting Data with db2split

You use the db2split program to split data across database partitions in a partitioned database. The program can be used in two ways:

- db2split can produce a new partitioning map, called an output partitioning map, that balances the data across the database partitions. An unbalanced or customized partitioning map can be created apart from db2split.
- You can have db2split with a partitioning map (balanced or unbalanced) to divide the data into separate files, one for each database partition. The input partitioning map is optional if created by db2split or required if customized by you.

To process data with db2split, the data must be a form eligible for use by the LOAD utility (except for IXF) data). Date, Time, and Timestamp data must be in the format:

```
DATE      YYYY-MM-DD
TIME      HH.MM.SS
TIMESTAMP YYYY-MM-DD-HH.MM.SS.XXXXXX
```

For more information on the LOAD utility, see “Using the LOAD Utility” on page 141.

The db2split program requires the data file, a configuration file and, optionally, an input partitioning map. (If a customized partitioning map was created, it must be used.) The configuration file contains such information as the name of the input file, the position and length of the partitioning key, and the name of the log file.

The partitioning map file must be generated for the set of nodes for your database (or particular nodegroup, if you created one). A partitioning map file is created when you create a nodegroup, or you can use db2split to create one, or you can use the db2gpmap program to obtain one from the system catalogs and specify that the db2split use it. The db2split program appends each row (or record) to the output file destined for the database partition at which this row should reside, as indicated by the partitioning map.

The db2split program supports codepage conversion. There are two parameters used with the program to specify the source codepage and the target codepage.

Using db2split

The db2split program is used in the following way:

```
db2split [option filename]
```

The options can be one of the following:

-c configuration file name

-d distribution file name

-i date input file name

-o output file prefix

-h help message

By default, db2split reads the db2split.cfg configuration file. When using the -c option with a file name, the file name is assumed to contain all of the input information for db2split. The input information is discussed in the section, "db2split Parameters" on page 841. The other options can be used to override the file names specified in the configuration file.

Note: You should always review and edit the db2split.cfg file before running db2split.

Populating a Table in a New Table Space

If you are populating a table in a new table space that does not yet contain data, do the following:

1. Execute db2split in analyze mode (to create a partitioning map), specifying a representative-sized data file for the largest table that will be in the table space. Analyze mode is established by setting the RunType parameter to ANALYZE in the configuration file.
2. Issue the REDISTRIBUTE NODEGROUP command, specifying the partitioning map created by db2split as the target partitioning map. This will put the partitioning map in the SYSPARTITIONMAPS catalog table.
3. Run db2split again, and have it partition the data according to the partitioning map that was just created. Partitioning is controlled by setting the RunType parameter to PARTITION in the configuration file.
4. Create the table in the table space with the CREATE TABLE statement.
5. Load the table on each database partition using the corresponding data files created during step 2. (The Load utility will check the partitioning map in the catalog table against the information in the data files that were split.)
6. Repeat steps three to five to split and load any other tables in the table space. When you do this, re-use the partitioning map that you created when you split the data of the first table.

Populating a Table in an Existing Table Space

If you are populating a table in an existing table space, it is usually better to have db2split use the existing partitioning map for the nodegroup, rather than create a new one. The reason is that db2split will create the best distribution for the table, but not necessarily for the table space. In this situation, you should use the db2gpmap program to obtain a copy of the partitioning map from the catalog partitioned database server and have db2split use this file to partition the data.

db2split Parameters

Following are the parameters that you set in the `db2split` configuration file:

Parameter	Description
Release	The release level of this program. When this program is used as part of DB2 Universal Database, it should be set to "V5.0." In all other cases, the release will be assumed to be DB2 Parallel Edition V1.
InFile	The input file name of the data you want partitioned. `db2split` recognizes `stdin` as a correct input file name and will receive input from it. If this field is not specified, `stdin` is used.
RecLen	The record length of the input data file. There are different meanings depending on the type of data file. For delimited data (file type DEL), this parameter is ignored. The record can be any length. For binary numeric data or packed decimal data (file type BIN or PACK), the exact record length deducted by 1 is used (for backward compatibility), and it has to be less than 32K in length. For positional ASCII data (file type ASC), and where each record has the same fixed length, specify the actual record length deducted by one. Again, it has to be less than 32K in length. For positional ASCII data (file type ASC), and where each record has a variable length, and each record is delimited with a line-feed character, it can be set to zero, and the `db2split` program will distinguish records by the new-line character. **Note:** The line-feed character in EBCDIC data is X'25'.
FileType	The data type of the input data file. Valid values are: **ASC** Positional ASCII file **DEL** Delimited ASCII file Each record must be delimited by a line-feed character. **BIN** Binary numeric data file. All numeric columns in the data file must be in binary format. Supported binary numeric data types include: Integer (4 bytes), Small Integer (2 bytes), Float (4 bytes), Double (8 bytes). Each record must be the same fixed length and may not be delimited by a new-line character. **PACK** Packed decimal data file. All decimal columns in the data file must be in the packed decimal format. Each record must be the same fixed length.

Note: A data file can be any combination of ASC, BIN, or PACK types. The `db2split` configuration file can contain two declarations of the `FileType` parameter: one for BIN and one for PACK. If both BIN and PACK are specified in two declarations of the `FileType` parameter, then all numeric columns must be in binary form, and all decimal columns must be in packed decimal form.

Nodes The database partitions on which the table is to be created. (If no partitioning map is provided, the program uses this parameter to generate one.) You can specify node numbers separately and as a range. Each separate number or range (except for the last) must be followed by a comma (,). For example, *Nodes*=(0,1-3,7-9,11,13-15).

Note: Always use the *Mapfili* parameter when the partitioning map is customized instead and not the default partitioning map. In the case where the partitioning map is the default, use this parameter or the *Mapfili* parameter, but not both.

OutputNodes The database partitions for which output files are to be created. The valid range is from 0 to 999.

OutputNodes must be a subset of *Nodes*.

If this parameter is not specified, output files are created for all database partitions.

OutputType If *OutputNodes* has only one member, use *OutputType* to specify whether the output should be written to a file (w) or piped to `stdout` (s).

The default is `stdout`.

If *OutPutNodes* has more than one member, *OutPutType* is ignored.

MapFili The filename of the input partitioning map.

Always use this parameter if the partitioning map is customized otherwise only use this parameter if you do not specify *Nodes*. In an analysis run, you can obtain the set of database partitions that is used to construct the output map from the input partitioning map. An analysis run is established by the `RunType` parameter described later in this list. Use this parameter or the `Nodes` parameter but not both to provide a partitioning map to the `db2split` program.

MapFilo The filename of the output partitioning map file.

This parameter is only meaningful during an analysis run.

DistFile The filename of the output distribution file.

This file is always written, and can be used as input by the data redistribution utility. The default name is `DISTFILE`.

LogFile The filename of the log file.

If this parameter is defined, all data is written to the specified file. If this parameter is not specified, output from the program is printed to the standard error device. After the filename, specify the mode:

w Open for write, truncate file to 0. This is the default.

a Open for write, append to the end.

OutFile The prefix of the output file.

`db2split` appends a 3-digit suffix (`000..999`) to the end of the prefix to generate the output file name if the *Release* parameter is "V5.0." Otherwise, `db2split` appends a 5-digit suffix (`00000...00999`) to the end of the prefix to generate the output file name.

The output files are named "`prefix suffix`." If the *OutFile* parameter is not specified, the default output filename prefix is `NOD.`

CDelimiter The column delimiter, which is used for DEL input files.

If `FileType` was not specified, this parameter can be used to determine if the datafile is ASC or DEL. If this parameter is not specified then the data file is an ASC file; otherwise, the data file is a DEL file. If specified, this parameter can be any character except line-feed, space, binary zero, or carriage-return.

SDelimiter The string delimiter.

This parameter is only meaningful with DEL files.

By default, the string delimiter is a double quotation mark (") and can be any character except line-feed, space, binary zero, carriage-return, or a period sign (.).

DecPt The decimal point.

This parameter is only meaningful with DEL files.

By default, the decimal point is a period (.) and can be any character except line-feed, space, carriage-return, or binary zero.

Note: `CDelimiter`, `SDelimiter`, and `DecPt` are all mutually exclusive.

Also, they have to be less than X'40' if the codepage of the data file is a double-byte character set (DBCS), Mixed, or EUC codepage. They cannot be shift-in (SI) or shift-out (SO) characters if the codepage of the data file is EBCDIC Mixed codepage.

Finally, you can specify delimiters in hexidecimal format. For example, you can use X'4F' or 0X'3A'.

RunType The type of run you want. Valid values are:

ANALYZE Produce the customized partitioning map

PARTITION Split the data

Check_Level The possible values are:

CHECK The program checks for the truncation of records at output. It also checks if the record is empty or not at input if the record length is less than 32K.

NOCHECK The program does not check for those things mentioned in CHECK.

Partition The partitioning key. The argument for this parameter has six fields, each field separated by commas:

- The column name that is used in the log file. This should be the same as the column name in the table.
- The cardinal value (starting with 1) of the partitioning field in each record. This is only valid for DEL data.
- The data offset for the start of the partitioning key (it starts at column 1). This is only valid for ASC data.
- The length of the partitioning key. This is only valid for ASC data.

  Note: With CHARACTER, FOR_BIT_DATA, FOR_BIT_VARCHAR, and VARCHAR delimited data, you must specify this field. The length should be equal to the corresponding column length in the database table.
- The null indicator. One of the following:

  N Null data is allowed.

  NN Not Null. The data must not be null.

  NNWD Not Null with Default. This is processed the same as NN.
- The type of data conversion for hashing into the partition index. One of the following:

  SMALLINT
  The same as integer conversion

  INTEGER
  The data is converted to a 4-byte integer

  FLOAT
  The data is a 4 byte float and is only valid with a BIN data file.

  DOUBLE
  The data is a 8 byte float and is only valid with a BIN data file.

  CHARACTER or CHAR
  Fixed-length character data

VARCHAR
: Variable-length character data

FOR_BIT_CHAR
: Fixed-length FOR_BIT character data

FOR_BIT_VARCHAR
: Variable-length FOR_BIT character data

DECIMAL(*x*,*y*)
: This converts to a packed decimal number, where *x* is the scale, and *y* is the fractional

DATE
: The data is converted to an internal format

TIME
: The data is converted to an internal format

TIMESTAMP
: The data is converted to an internal format

One *Partition* statement is used for each column of the partitioning key (from high order to low order). An ASC example is as follows:

```
Partition=cntl_no,,1,8,N,DECIMAL(8,0)
```

Trace Put trace information for a specified number of records into the log file.

header This parameter guides `db2split` to generate the header information or not.

If the parameter is YES, header information is generated for all splitting tables. Otherwise, no header information is generated for the splitting files.

The default for this parameter is YES.

DATA_CODEPAGE The codepage of the input data file.

The codepage must be a database manager convertible codepage. If it is not provided, it is assigned the codepage number for the database if it is specified; otherwise, it is assigned the codepage number of the application.

DB_CODEPAGE The codepage of the database, where the table is defined.

The codepage must be a database manager supported codepage. If it is not provided, it is assigned the codepage number of the input data file; otherwise, it is assigned the codepage number of the application.

NewLine This parameter allows for checking of expected and actual record lengths in the data file by `db2split`.

This parameter is only meaningful for an ASC file.

The parameter values can be YES or NO. The default for this parameter is NO.

If the parameter is YES, and the RecLen parameter is not zero, db2split recognizes records by the new-line delimiter. Then it compares the actual record lengths with the expected record length. If there is no match, an error is returned for that record.

If the parameter is NO, no checking is done.

Example Data File for db2split

This section provides an example of a data file and the table into which data is to be loaded. The following sections provide an example of the configuration file that you would use, and an explanation of how the file is set up.

Assume that you have a delimited data file called MYDATA, which is as follows:

```
25,125,dog,123.45,1984-12-15,12.00.23,1984-12-14-11.11.59.000000
,,cat,12.34,,,1982-12-15-11.11.25.001200
213,424,bird,56.345,,,
```

Also assume that you want your table to be distributed on 3 nodes (4,7,8), and that you create it with the following definition:

```
db2 CREATE NODEGROUP MyNodeGroup ON NODES (4,7,8)

db2 CREATE TABLESPACE MyTableSpace IN NODEGROUP MyNodeGroup

db2 CREATE TABLE MyTable (col1 INTEGER,
                          col2 SMALLINT,
                          col3 VARCHAR(5),
                          col4 DECIMAL,
                          col5 DATE,
                          col6 TIME,
                          col7 TIMESTAMP)
        IN MyTableSpace
        PARTITIONING KEY (col7,col1,col3,col4,
                          col2,col5)
```

AIX Configuration File

```
Infile=MYDATA
;
Nodes=(4,7,8)
OutputNodes=(4,7,8)
; MapFili=MyInputMap
MapFilo=MyOutputMap
OutFile=MyOutput
RunType= partition
DistFile=DISTRIBUTION
LogFile=MyLog
CDelimiter=,
```

```
SDelimiter="
Partition=col7,7,,,N,timestamp
Partition=col1,1,,,N,integer
Partition=col3,3,,5,N,character
Partition=col4,4,,,N,decimal(6,3)
Partition=col2,2,,,N,smallint
Partition=col5,5,,,N,date
Trace=20
FileType=DEL
```

Notes:

1. The *Nodes* field should be consistent with the table nodegroup.

   If the *MyNodeGroup* definition was:

   ```
   db2 CREATE NODEGROUP MyNodeGroup ON NODES (0,1,2)
   ```

   Then the *Nodes* parameter could be specified as `Nodes=(0-2)` in the configuration file. `db2split` generates a partitioning map with repetitive sequences of the elements specified in this parameter.

   You would not specify this field if you were specifying *MapFili*.

2. In this situation, the *OutputNodes* field is also optional. By default, the output nodes have the same members as the *Nodes* field.

   If you specify `OutputNodes=(4,8)`, the program writes the results of database partitions 4 and 8 to the output data file as defined in *OutFile*. Consequently, you will have two output files: `MyOutput.004` and `MyOutput.008`.

   If you want only the output data for database partition 7, then specify `OutputNodes=(7)` (the parentheses are required). In this situation, *OutputNodes* has only one member. The result will be written to `stdout`. If you want to direct the output to the file `MyOutput.007` rather than `stdout`, set *OutputType* to w:

   ```
   OutFile=MyOutput
   OutputType=w
   ```

3. For partitioning keys, the column name field of character data can be anything you want. Normally, this field is used to identify the corresponding column name from the table.

   The length field of character data type must be consistent with that of the database table. In this situation, the table is created with a column length of 5. As a result, the length field of the partitioning key `col3` is set to 5 as well.

   The first decimal record (123.45) suggests that `Partition=col4,4,,,N,decimal(5,2)` should be specified. The third record, however, is 56.345, which means that the precision has to be increased to 3 and the total length to 6, as follows:

   ```
   Partition=col4,4,,,N,decimal(6,3)
   ```

Getting a Partitioning Map with db2gpmap

If you have already set up a database and defined the nodegroups for it, the db2gpmap tool gets the partitioning map for the database table or the nodegroup from the catalog partitioned database server.

When running db2gpmap, you must set the $DB2DBDFT environment variable to correspond to the database that you want to access, or use the command line options to override $DB2DBDFT.

For additional information, issue the following command:

```
db2gpmap -h
```

The db2gpmap program is used in the following way:

```
db2gpmap [option ( parameter )]
```

The options can be one or more of the following:

-d database name (default is sample)

-m map filename (default is db2split.map)

-g nodegroup name (default is IBMDEFAULTGROUP)

-t table name

-h help message

Before using db2gpmap the database manager must be started and the db2gpmap.bnd must be bound to the database. If not already bound to the database before you use this program, the program will attempt to bind the file on its own.

Running db2split

After creating your configuration and partitioning map files, you must decide where you want to run db2split. Apart from processor power, the other main consideration is the amount of disk space you have available to hold the partitioned files. You need at least the same amount as that taken up by the source data.

The following are the steps to run db2split:

1. Create the configuration file (using db2split.cfg as a sample) and the partitioning map file, if you have not done so already.
2. Execute db2split as follows:

   ```
   db2split -c <configfile>
   ```

When you split the files, db2split writes header information to each data file that it creates. For details, see “db2split Header Information” on page 849.

db2split Header Information

When you use `db2split` to split data, the program writes header information to each data file that it creates. The Load utility uses this information to ensure that the data goes to the correct location. The information is as follows:

- The first line is the node number, which is a digit.
- The next block of entries is the 4096 entries of the partitioning map, which is separated by a blank record both above and below the block.
- The next line is the separator `SQL_HEADER_DATA_SEPARATOR`.
- The next line is the number of partitioning keys, which is a digit.
- The next block of entries is made of as many lines as there are partitioning columns. Each record in this block is as follows:

  ```
  Key: Name, Type, TypeLength, Start, Length
  ```

- The next line is the separator `SQL_HEADER_DATA_SEPARATOR`.

Appendix O. Supplemental AutoLoader Information

In a partitioned database environment, you may want to load data across all the partitions at the same time. You can use the AutoLoader utility to do this. This appendix supplements the information in the “Using the AutoLoader Utility” on page 153 section. This appendix is specific to all **AIX systems** only.

The table of contents:

Introduction

The `AutoLoader` is a tool that can:

1. TRANSFER data from one system (like MVS) to another system (like UNIX).
2. SPLIT (or partition) that data in parallel.
3. LOAD the data simultaneously on the corresponding database partitions.

The `AutoLoader` may be run in one of four modes:

1. SPLIT_AND_LOAD. Data is split (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
2. SPLIT_ONLY. Data is split (perhaps in parallel) and the output from the splitters is written to files in the `SPLIT_FILE_LOCATION` or in the AutoLoader current working directory.
3. LOAD_ONLY. Data is expected to be already split and contained in files with the following naming convention:

   ```
   filename.xxx
   ```

   where `xxx` is the node number. The split process is skipped and data is loaded simultaneously on the corresponding database partitions. The assumption is made that `filename.xxx` is found in the `SPLIT_FILE_LOCATION` or in the current AutoLoader working directory.
4. ANALYZE. An optimal partitioning map with even distribution across all database partitions is generated. It is recommended that a data file with a fairly large number of records from the largest table in the database be used for this analysis.

Before you continue, you should be familiar with `db2split` and the load process. Refer to Appendix N, “Splitting Data with db2split” on page 839 and “Using the LOAD Utility” on page 141 for more information on these topics.

Files

`AutoLoader` requires the following files:

Table 106. AutoLoader Files

FILE	DESCRIPTION	LOCATION
db2autold	The main AutoLoader driver	/sqllib/misc
autoload.cfg	A sample configuration file	/sqllib/samples/autoloader
db2psplit	Executable	/sqllib/misc
db2pread	Executable	/sqllib/misc
db2pmove	Shell script	/sqllib/misc
db2atldm	Shell script	/sqllib/misc

Setup for AutoLoader

Before running `AutoLoader`:

1. You must first create a temporary working directory and copy the file `/sqllib/samples/autoloader/autoloader.cfg` into this temporary directory. This directory is where you should invoke AutoLoader and it has to be NFS-accessible (read and write) to all participating split and load database partitions.
2. Modify the `autoloader.cfg` file according to the instructions in the sample.
3. The shell program `ksh` is assumed to be in the `/bin` directory. If you find it in some other directory, change the first line `#!/bin/ksh` of all the shell scripts to `#!your_ksh_directory/ksh`. You can test this using the `which ksh` or the `whence -v ksh` command. Also, since the shell scripts use `sed`, `mkdir`, `awk`, `cut`, `grep`, `egrep`, `tr`, `ftp`, `cat`, `lsuser`, and `rsh` commands, make sure you have all these programs installed properly. And program `mknod` is assumed to be in `/etc` directory. If it is not, please assign a proper value to MKNOD_DIR variable.
4. If data file is remote, put the following information in the `$HOME/.netrc` file:

   ```
   machine machine_name login login_name password login_password
   ```

   where machine_name is the hostname of the source_system (i.e. the place we need to ftp from to get the data file.

   Ensure that the permissions of .netrc files are -rw------- If not, use the following command to change the permissions.

   ```
   $ chmod 600 .netrc
   ```

5. On each participating database partition (splitting and loading), there must be a not-nfs directory, which has identical path name and is writable. The default is assumed to be `/notnfs`. A sub-directory with the name `$USERID` is expected in the `notnfs` path as well. That is, the path is expected to include: `/notnfs/$USERID/`.

Usage

Syntax:

```
db2autold [-d] [-i] [-c config_file]
```

Flags:

-c Use the `config_file` as the configuration file for AutoLoader. The default is `autoloader.cfg`.

-d In case of abnormal exit from AutoLoader, it is necessary to run AutoLoader with this option to clean up all associated temporary directories, files, and hanging processes. Option `-c` is also needed to specify the configuration file for the unsuccessful AutoLoader session, which left garbage around.

-i In case the clean up in the end has to be interactive. By default, clean will be done without a prompt.

AutoLoader creates a log file called `autoload.log`. This contains the messages from the main AutoLoader script. Check it to ensure all pipes and temporary directories got set up correctly. It also creates a file called `load_log.XXX`. This contains messages from the load process on database partition XXX. As well, AutoLoader also creates a file called `splt_log.XXX`. This contains messages from the split process on database partition XXX.

Hints and Tips

1. Test with small amounts of data to get familiar with the tool before loading huge amounts.
2. If the input data is already sorted (or in some order) and you wish to maintain that order during the load process, then use only one database partition for splitting. Parallel splitting will not guarantee that the data will be loaded in the same order as it was received.
3. If LOBs are stored within separate files (that is, you are making use of the LOBSINFILE feature), then all directories containing LOB files should be made accessible to all the database partitions where loading is taking place.
4. AutoLoader will ignore the MESSAGES parameter in a LOAD command, and it will direct all messages from the LOAD command into the file load_log.XXX as described in “Usage” above.
5. AutoLoader will only choose one output database partition to collect statistics. The option RUN_STAT_NODE in the AutoLoader configuration file is provided for this purpose.
6. Multiple invocations of AutoLoader may be used to load data simultaneously into separate tables. However, make sure that:
   a. The tables reside on separate table spaces.
   b. All the AutoLoader's are invoked from separate directories.
   c. AutoLoader uses the datafile name to create some temporary pipes. Ensure that the file name is unique for the multiple AutoLoader's.

Troubleshooting

1. If it appears that db2autold is hanging, you can:
   - Check load_log.node_num files to get the progress of load
   - Check load_log.node_num files to get the progress of load on each database partition.
   - Check splt_log.node_num files to see the status of db2split processes on each splitting database partition. If things are going well and the TRACE parameter is set in the db2autold configuration file, there should be trace messages for a certain number of records in these log files.
   - Check db2psplit_msg, db2pread_msg.node_num, and db2pmove_msg.node_num files to see if there are any error messages.
   - If you do find errors that would suggest db2split is not doing anything, or that db2psplit or db2pread or db2pmove encountered errors, then you should interrupt the current db2autold.
2. If there is a failed or interrupted db2autold, the first thing you have to do is to clean up the working environment by issuing:

   ```
   db2autold -c your_cfg_file -d
   ```

   Make sure your_cfg_file is the same configuration file that the failed or interrupted db2autold used.
3. If db2autold still fails you can try the following to diagnose the problem:
   - Change MODE in your db2autold configuration file to SPLIT_ONLY. Run db2autold again. Check the split data files to see if there is anything abnormal in them.
   - If the split files look correct, then try to load one of those split files manually on the right database partition.
   - If the data loads OK, then there might be some db2autold functional problems or database system problems. Please contact your IBM service representative.

Appendix P. Issuing Commands to Multiple Database Partitions

In a partitioned database environment, you may want to issue commands to be run on all database hosts or just on some database partitions. You can do so using the rah shell script or the db2_all shell script. This appendix gives you an overview of these shell scripts. This appendix is specific to all **UNIX systems** only.

The rah command allows you to issue commands that you wish to run at all hosts in a list. If you wish the commands to run at all *partitions* in a list, you run the db2_all shell script. Your login shell can be a kornshell or any other shell; however, there are differences in the way the different shells handle commands containing special characters.

Commands

You can run the commands sequentially at one database partition after another or you can run the commands in parallel. If you run the commands in parallel, you can either choose to have the output sent to a buffer and collected for display (the default behaviour) or the output can be displayed at the node where the command is issued.

To use rah, type:

```
rah command
```

To use db2_all, type:

```
db2_all command
```

To get help on rah syntax, type

```
rah "?"
```

The command is run using rshell to send the command to each host. The command can be almost anything which you could type at an interactive shell prompt, including, for example, multiple commands to be run in sequence. You separate multiple commands using a semicolon (;). Do not use the semicolon following the last command.

The following example shows how to use the db2_all command to load data on several partitions specified in the db2nodes.cfg file. Because a semicolon was placed inside the double quotes, the request will run concurrently. The messages from the load command will be sent back to the partitions where the db2_all command is run. The output messages will be placed into an output file called load.test1.out.

```
db2_all ";$HOME/load.test1"  |tee load.test1.out
```

Defining synonyms

The following commands should be defined as synonyms for rah.

Synonym	Function
db2_all	Runs the command on all partitions you specify

db2_kill Abruptly stops all processes being run on multiple partitions and cleans up all ipcs on all partitions. This command renders your databases inconsistent. Do NOT issue this command except under direction from IBM service.

db2_call_stack Causes all processes running on all partitions to write call traceback to syslog

These commands execute `rah` with certain implicit settings such as:

- Run in parallel at all hosts
- Buffer command output in /tmp/$USER/db2_kill, /tmp/$USER/db2_call_stack respectively.

Specifying the Command to be Run

You may specify the command:

- From the command-line as the parameter
- In response to the prompt if you don't specify any parameter.

You should use the prompt method if the command contains the following special characters:

```
| & ; < > ( ) ` { } [ ] unsubstituted $
```

The command will be added to your shell history just as if you typed it at the shell prompt. If you specify the command as the parameter on the command-line, you must enclose it in double-quotes if it contains any of the special characters just listed.

All special characters in the command can be entered normally (without being enclosed in quotes except for \). If you need to include a \ in your command, you must type two backslashes (\\).

Note: For non-ksh users, all special characters in the command can be entered normally (without being enclosed in quotes except for ` " \ unsubstituted $, and the single-quote (')). If you need to include one of these characters in your command, you must precede them by three backslashes (\\\). For example, if you need to include a \ in your command, you must type four backslashes (\\\\).

If you need to include a double quote (") in your command, you must precede it by three backslashes, for example, \\\". You cannot include a single-quote (') in your command unless your shell provides some way of entering a single-quote inside a singly-quoted string.

Running Commands in Parallel

By default, the command is run sequentially at each host, but you can specify to run the commands in parallel using background rshells by prefixing the command with certain prefix sequences. If the rshell is run in the background then each command puts the output in a buffer file at its remote host. This process retrieves the output in two pieces:

1. After the remote command completes
2. After the rshell terminates which may be later if some processes are still running.

The name of the buffer file is /tmp/$USER/rahout by default, but it can be specified by the environment variables $RAHBUFDIR/$RAHBUFNAME.

When you specify that you want the commands to be run concurrently, by default, this script prepends an additional command to the command sent to all hosts to check that $RAHBUFDIR and $RAHBUFNAME are usable for the buffer file. It creates $RAHBUFDIR. To suppress this, export an environment variable RAHCHECKBUF="no". You can do this to save time if you know the directory exists and is usable.

Before using `rah` to run a command concurrently at multiple hosts, ensure that:

- A directory `/tmp/$USER` exists for your userid at each host. To create a directory if one does not already exist, run

  ```
  rah ")mkdir /tmp/$USER"
  ```

- Add the following line to your .kshrc or .profile, and also type it into your current session:

  ```
  export RAHCHECKBUF=no
  ```

- Ensure that each host id at which you run the remote command has an entry in its `.rhosts` file for the id which runs `rah`; and the id which runs `rah` has an entry in its `.rhosts` file for each host id at which you run the remote command.

Monitoring rah Processes

While any remote commands are still running or buffered output is still being accumulated, processes started by `rah` monitor activity to:

- Write messages to the terminal indicating which commands have not been run
- Retrieve buffered output.

The informative messages are written at an interval controlled by the environment variable RAHWAITTIME. See the help information for details on on how specify this. All informative messages can be completely suppressed by exporting RAHWAITTIME=0.

The primary monitoring process is a command whose command-name (as shown by ps command) is rahwait>or. The first informative message tells you the pid (process id) of this process. All other monitoring processes will appear as ksh commands running the rah script (or name of symbolic link). If you wish, you can stop all monitoring processes by the command:

```
kill <pid>
```

where <pid> is the process id of the primary monitoring process. Do not specify a signal number. Leave the default of 15. This will not affect the remote commands at all, but will prevent automatic display of buffered output. Note that there may be two or more different sets of monitoring processes executing at different times during the life of a single execution of `rah`. However, if at any time you stop the current set, then no more will be started.

If your regular login shell is not a kornshell (for example /bin/ksh), you can use `rah` but there are some slightly different rules on how to enter commands containing the following special characters:

```
" ` unsubstituted $ '
```

For more information, type `rah "?"`. Also, in a UNIX-based environment, if the login shell at the id which executes the remote commands is not a kornshell, then the login shell at the id which executes `rah` must also not be a kornshell. (`rah` makes the decision as to whether the remote id's shell is a kornshell based on the local id). The shell must not perform any substitution or special processing on a string enclosed in single quote marks. It must leave it exactly as is.

Prefix Sequences

A prefix sequence is one or more special characters. Type one or more prefix sequences immediately preceding the characters of the command without any intervening blanks. If you want to specify more than one sequence you can type them in any order, but characters within any multi-character sequence must be typed in order. If you type any prefix sequences, you must enclose the entire command including the prefix sequences in double quotes as shown in the following example:

```
rah "};ps -F pid,ppid,etime,args -u $USER"
```

The prefix sequences are:

Sequence	Purpose
|	Runs the commands in sequence using background rshells.
|&	Runs the commands in sequence using background rshells AND terminates the script after all remote commands have completed even if some rshells are still running (which may be later if for example, child processes are still running). In this case, the script starts a separate background process to retrieve any remote output generated after command termination and writes it back to the originating terminal. For example, if the command is db2start, the rshell will persist until db2stop. All output after the completion of db2start will be retrieved by the process. **Note:** Specifying & degrades performance since more rsh commands are required.
||	Runs the commands in parallel using background rshells.
||&	Runs the commands in parallel using background rshells and terminates the script after all remote commands have completed as described for the |& case above. **Note:** Specifying & degrades performance since more rsh commands are required.
;	Same as ||& above. It is an alternative shorter form. NOTE - specifying ; degrades performance relative to || since more rsh commands are required.
]	Prepends dot-execution of user's profile before executing command.
}	Prepends dot-execution of file named in $RAHENV (probably .kshrc) before executing command.

]} Prepends dot-execution of user's profile followed by execution of file named in $RAHENV (probably .kshrc) before executing command.

) Suppresses execution of user's profile and of file named in $RAHENV.

' Echoes the command invocation to terminal.

< Sends to all hosts except this host.

<<–**nnn**< Sends to all-but-partition nnn (all partitions in db2nodes.cfg except partition number nnn, see note below).

<<+**nnn**< Sends to only partition nnn (the partition in db2nodes.cfg whose partition number is nnn, see note below).

` Runs the remote command as a daemon, i.e. in background with `stdin`, `stdout` and `stderr` all closed. This option is valid only when using background rshells, i.e. only in a prefix sequence which also includes | or ;. It allows the rshells to complete much sooner (as soon as the remote command has been initiated). If you specify this prefix character on the rah command line, then either enclose the command in single-quotes or enclose the command in double-quotes AND precede the ` by a \. For example,

```
rah ';`mydaemon'
```

or

```
rah ";\`mydaemon"
```

When the command is run as a daemon, `rah` will never wait for any output to be returned.

> Substitutes occurrences of <> by hostname.

" Substitutes occurrences of () by host index, and substitutes occurrences of # by partition number.

Note: When " is specified, duplicates are not eliminated from the list of hosts (see below).

When using `<<–nnn<` and `<<+nnn<` prefix sequences, nnn is any 1, 2 or 3 digit decimal partition number which must match the first token in one line of db2nodes.cfg.

Note: Prefix sequences are considered to be part of the command. If you specify a prefix sequence as part of a command, you must enclose the entire command including the prefix sequences in double quotes.

Specifying the List of Hosts

By default the list of hosts is taken from (in order of search):

- <db2instance_home_dir>/sqllib/db2nodes.cfg
- $HOME/sqllib/db2nodes.cfg

You can override this by:

- Specifying the pathname of the file containing the list of hosts by exporting the environment variable `RAHOSTFILE`.

- Specifying the list explicitly, as a string of names, by exporting the environment variable `RAHOSTLIST`. If both of these environment variables are exported, then RAHOSTLIST takes precedence.

Eliminating Duplicate Entries from the Host List

If you are running DB2 Extended Enterprise Edition with multiple database partitions on one host machine, your db2nodes.cfg will contain multiple entries for the same host. In this case, the `rah` command needs to know whether you want the command to be executed once only on each distinct host or once for each entry in db2nodes.cfg (partitions). Use the `rah` shellscript to choose hosts. Use the `db2_all` shellscript to choose partitions.

If you specify hosts, then `rah` will normally eliminate duplicates from the host list, with the following exceptions:

- If you specify partitions, then db2_all will prepend to your command, the assignment

  ```
  export DB2NODE=nnn
  ```

  where nnn is the nodenumber taken from the first word of the corresponding line in db2nodes.cfg, so that the command will be routed to the desired partition.

When specifying partitions, you can restrict the list of partitions to either all nodes except one single partition or one single partition using the `<<-nnn<` and `<<+nnn<` prefix sequences. You may want to do this if you wish to run a command at the Catalog partition first, and when that has completed, run the same command at all other partitions possibly in parallel. This is usually required when running the `db2 restart database` command. You will need to know the partition number of the Catalog partition to do this.

If you execute this using the `rah` shellscript, duplicate entries are eliminated from the list of hosts. However if you specify the " prefix, then duplicates are not eliminated since it is assumed that use of the " prefix implies sending to each partition rather than each host.

Controlling the Shell Script

You can use environment variables to control the shell script.

Table 107 (Page 1 of 2).

Name	Meaning	Default
$RAHBUFDIR	directory for buffer	/tmp/$USER
$RAHBUFNAME	filename for buffer	rahout
$RAHOSTFILE	pathname of file containing list of hosts	<db2instance_home_dir>/sqllib/db2nodes.cfg
$RAHOSTLIST	list of hosts as a string	extracted from $RAHOSTFILE
$RAHCHECKBUF	if set to "no", bypass checks	not set

Table 107 (Page 2 of 2).

Name	Meaning	Default
$RAHSLEEPTIME	time in secs this script will wait for initial output from commands run in parallel	86400 sec for db2_kill, 200 sec for all other
$RAHWAITTIME	interval in secs between successive checks that remote jobs are still running and "rah: waiting for <pid> ..." messages. Specify any positive integer. Prefix value with a leading zero to suppress messages for example, export RAHWAITTIME=045. (Note - not necessary to specify a low value as rah does not rely on these checks to detect job completion).	45 sec
$RAHENV	specifies filename to be executed if $RAHDOTFILES=E or K or PE or B	$ENV
$RAHUSER	userid under which the remote command is to be run	$USER

Note: For $RAHENV, the value of $RAHENV where `rah` is run is used, not the value (if any) set by the remote shell.

$RAHDOTFILES

Following are the .files that are run if no prefix sequence is specified:

P .profile
E File named in $RAHENV (probably .kshrc)
K Same as E
PE profile followed by file named in $RAHENV (probably .kshrc)
B Same as PE
N None (or Neither)

Note: If your login shell is not kornshell, any dot files which you specify to be executed will be executed in a kornshell process and so must conform to kornshell syntax. So, for example, if your login shell is csh, then to have your .cshrc environment set up for commands executed by rah, you should either create a kornshell $HOME/.profile equivalent to your `.cshrc` and specify in your $HOME/.cshrc

```
setenv RAHDOTFILES P
```

or you should create a kornshell $HOME/.kshrc equivalent to your .cshrc and specify in your $HOME/.cshrc

```
setenv RAHDOTFILES E
setenv RAHENV $HOME/.kshrc
```

Also, it is essential that your .cshrc does not write to stdout if there is no tty (as when invoked by rsh). You can ensure this by enclosing any lines which write to stdout by, for example,

```
if { tty -s } then echo "executed .cshrc";
endif
```

Determining problems with rah:

Here are suggestions on how to handle some problems that you may encounter when you are running `rah`:

1. rah hangs (or takes a very long time)

   This problem may be caused because:

   - rah has determined that it needs to buffer output, and you did not export RAHCHECKBUF=no therefore before running your command `rah` has sent a command to all hosts to check the existence of the buffer directory and to create it if it doesn't exist.
   - One or more of the hosts where you are sending your command is not responding. The rsh command will eventually timeout but the timeout interval is quite long, usually about 60 seconds.

2. You have received messages such as:

   a. Login incorrect
   b. Permission denied

   Either one of the hosts does not have the id running `rah` correctly defined in its `.hosts` file or the id running rah does not have one of the hosts correctly defined in its `.rhosts`

3. When running commands in parallel using background rshells, although the commands run and complete within the expected elapsed time at the hosts, `rah` takes a long time to notice this and put up the shell prompt.

   The id running rah does not have one of the hosts correctly defined in its `.rhosts` file

4. Although `rah` runs fine when run from the shell command line, if you run `rah` remotely using rsh, for example,

   ```
   rsh somewher -l $USER db2_kill
   ```

   `rah` never completes.

   This is normal. `rah` starts background monitoring processes which continue to run after it has exited. Those processes will normally persist until all processes associated with the command you ran have themselves terminated. In the case of `db2_kill`, this means termination of all database managers. You can terminate the monitoring processes by finding the process whose command is rahwait>or and kill <process_id>. Do not specify a signal number - leave to default (to 15).

5. The output from `rah` is not displayed correctly, or `rah` incorrectly complains that $RAHBUFNAME does not exist, when multiple commands of `rah` were issued under the same $RAHUSER.

   This is because multiple concurrent executions of rah are trying to use the same buffer file (ie $RAHBUFDIR/$RAHBUFNAME) for buffering the outputs. To prevent this problem, use a different $RAHBUFNAME for each concurrent rah as, for example in the following ksh:

```
export RAHBUFNAME=rahout
rah ";$command_1" &
export RAHBUFNAME=rah2out
rah ";$command_2" &
```

or use a method which makes the shell choose a unique name automatically such as:

```
RAHBUFNAME=rahout.$$ db2_all "....."
```

Whatever method you use, you must ensure you clean up the buffer files at some point if disk space is limited. `rah` does not erase a buffer file at the end of execution, although it will erase and then re-use an existing one the next time you specify the same buffer file.

6. You entered

```
rah '"print from ()'
```

and received the message:

```
ksh: syntax error at line 1 : `(' unexpected
```

Prerequisites for the substitution of () and ## are:

- Use `db2_all`, not `rah`
- Ensure a RAHOSTFILE is used either by exporting RAHOSTFILE or by defaulting to your `~/sqllib/db2nodes.cfg` file. Without these prerequisites, `rah` will leave the () and ## as is. You get an error because the command `print from ()` is not valid.
- Performance tip when running commands in parallel:

  Use | rather than |& and use || rather than ||& or ; unless you truly need the function provided by & since specifying & requires more rsh commands and therefore degrades performance.

Appendix Q. Supporting High Availability Cluster Multi-Processing Configurations

DB2 UDB for AIX provides high availability failover support through the capabilities of IBM High Availability Cluster Multi-Processing for AIX (HACMP). This allows for the automatic transfer of workload from one processor to another should there be a hardware failure.

If you are looking for high availability options for platforms other than AIX, you could periodically review the IBM DB2 web site at http://software.ibm.com/data/db2/ As support of similar offerings become available, they will be announced at this site.

HACMP provides increased availability through clusters of processors which share resources such as disks or network access. If one processor fails then another in the cluster can substitute for the failed one.

There are three modes of failover support provided, a brief description of each mode and its application to DB2 follows. In each case we use the simple scenario of a two processor HACMP cluster.

Hot Standby

One processor is being actively used to run your DB2 instance and the second is in standby mode ready to take over the instance if there is an operating system or hardware failure involving the first processor.

Mutual Takeover

Both processors are either used to run separate DB2 instances, or one is use to run a DB2 instance while the other is used to run DB2 applications. If there is an operating system or hardware failure on one of the processors, the other processor takes over the tasks of the failing processor. Once the failover is complete, the remaining processor is doing the work of both processors.

Concurrent Access

Multiple processors can be used to scale to a single database instance using the DB2 Universal Database Extended Enterprise Edition product. This is done using a shared-nothing model and partitioning the data such that one or more partitions are running on each processor in the cluster. If an operating system or hardware failure occurs on one of the processors, then the other processor will take over the partitions of the failing processor. DB2 UDB Extended Enterprise Edition does not require the use of a Concurrent Resource Manager to provide redundancy. DB2 co-exists with the Concurrent Resource Manager, but does not require its capability. Redundancy is managed by using the previous two modes. The capabilities of this mode are only required by database managers with a shared architecture.

Each of the above configurations can be used to failover one or more partitions of a partitioned database. In addition, each can failover a complete instance of a single partition installation.

Hot Standby

The Hot Standby capability can be used to failover the entire instance of a single partition database or a partition of a partitioned database configuration. If one processor fails then another processor in the cluster can substitute for the failed processor by automatically transferring the instance. In order to achieve this, the database instance and the actual database must be accessible to both the primary and failover processor. This requires that the following installation and configuration tasks be performed:

- The DB2 installation path can either be on a path shared by both systems or on a non-shared filesystem. If using a non-shared file system the installation levels must be identical.
- The DB2 instance path, as with the installation path can either be on a shared filesystem or on a manually mirrored filesystem.
- Database and the associated containers must be on file systems (or devices) accessible to both systems.
- There are sample scripts which can be tailored to perform the failover tasks. Refer to the subsequent examples for more details on these scripts.
- For failover of a partition in a partitioned database configuration, the partition is restarted on the second processor: the failover script changes the db2nodes.cfg file to point to the failed partition on the new processor and starts the partition on that processor.
- When a failover occurs, the external communications addresses for supported communication protocols are transparently transferred as part of the failover procedure.

For detailed information on the actual installation requirements and instance creation, refer to *HACMP for AIX, Version 4.2: Installation Guide*, SC23-1940.

Examples

Each of the following examples has a sample script stored, on AIX-based installations, in `sqllib/sample/hacmp`.

Instance Failover

The first example of a hot standby failover scenario consists of a single two processor HACMP cluster running a single-partition database DB2 instance. Figure 73 on page 867 shows, at a high level, this configuration. This diagram is intended to depict the major elements of the cluster, not a complete configuration. For information on configuring your HACMP cluster, refer to “Additional HACMP Resources” on page 872.

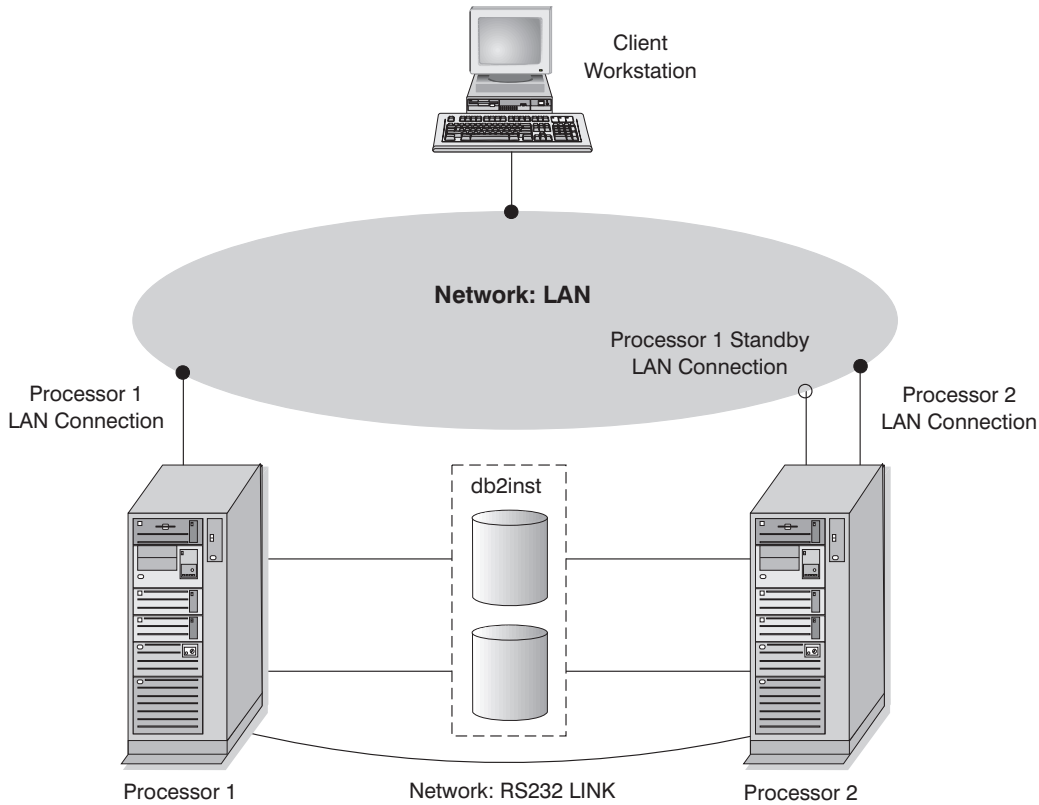


Figure 73. Instance Failover Example

Both processors have access to the installation directory, the instance directory, and the database directory. The database instance "db2inst" is being actively executed on processor 1, processor 2 is not active and is being used as a hot standby. A failure occurs on processor 1 and the instance is taken over by processor 2. Once the failover is complete both remote and local applications can access the database within instance "db2inst". The database will either have to be manually restarted; or, if AUTORESTART is on, the first connection to the database will cause the restart. In the sample script provided, it is assumed that AUTORESTART is off and the failover script performs the restart for the database. See “Overview of Recovery” on page 179 for additional information on AUTORESTART.

Sample script:

hacmp-s1.sh

Partition Failover

The second example is slightly more complex than that of a simple instance failover: In this example, we are actually using a partition of an instance as opposed to the entire instance. We will use the two processor HACMP cluster as in the previous example, but

the machine will represent one of the partitions of a partitioned database server. Processor 1 will be running a single partition of the overall configuration and processor 2 will be used as the failover processor. When processor 1 fails, the partition is restarted on the second processor. The failover updates the db2nodes.cfg file, pointing the partition to processor 2's hostname and netname, and then restarting the partition at the new processor. Once complete, all other partitions forward the requests targeted for this partition to processor 2.

The following is a portion of the db2nodes.cfg file before and after the failover. In this example, node number 2 is running on processor 1 of the HACMP machine which has a hostname of “node201” and the netname is the same. After the failover, node number 2 is running on processor 2 of the HACMP machine which has a hostname of “node202” and the netname is the same. The failover script will execute the command between the before and after definitions.

```
Before:
        1 node101 0 node101
        2 node201 0 node201    <= HACMP
        3 node301 0 node301

        db2start nodenum 2 restart node202 port 0 netname node202

After:
        1 node101 0 node101
        2 node202 0 node202    <= HACMP
        3 node301 0 node301
```

Sample script:

hacmp-s2.sh

Multiple Logical Node Failover

A more complex variation of the previous example involves the failover of multiple logical nodes from one processor to another. Again, we are using the same two processor HACMP cluster configuration as above. However, in this scenario, processor 1 is running 3 logical partitions. The setup is the same as that for the simple partition failover scenario, but in this case when processor 1 fails each of the logical partitions must be started on processor 2. Each logical partition must be started in the order that it is defined in the db2nodes.cfg file: the logical partition with port number 0 must always be started first.

The following is a portion of a db2nodes.cfg file which has 3 logical partitions defined on processor one of the two processor HACMP cluster scenario. The example uses the same hostnames and netnames as the previous example.

```
Before:
        1 node101 0 node101
        2 node201 0 node201    <= HACMP
        3 node201 1 node201    <= HACMP
        4 node201 2 node201    <= HACMP
        5 node301 0 node301

        db2start nodenum 2 restart node202 port 0 netname node202
        db2start nodenum 3 restart node202 port 1 netname node202
        db2start nodenum 4 restart node202 port 2 netname node202

After:
        1 node101 0 node101
        2 node202 0 node202    <= HACMP
        3 node202 1 node202    <= HACMP
        4 node202 2 node202    <= HACMP
        5 node301 0 node301
```

Sample script:

hacmp-s3.sh

Mutual Takeover

DB2's exploitation of the mutual takeover mode has the same basic characteristics as that for the hot standby mode. In this mode, one processor can failover the single-partition database instance, or the partitions of a partitioned database, of a failed processor while running another instance or other partitions of a partitioned database configuration. As with the hot standby configuration, the installation path, the instance directory, and the database must be mutually accessible by each processor which may be involved in failover processing. The installation and instance paths can either be on a shared filesystem or mirrored on separate filesystems.

When utilizing the mutual takeover mechanism, for instance failover, the instances must be defined in such a manner that both instances can be run on the same processor at the same time. For detailed information on the actual installation requirements and instance creation, refer to *HACMP for AIX, Version 4.2: Installation Guide*, SC23-1940.

Examples

Each of the following examples has a sample script stored, on AIX-based installations, in `sqllib/sample/hacmp`.

Mutual DB2 Instance Failover

In order to illustrate a mutual instance failover, we will use the simple case of a HACMP system with two processors known as “node10” and “node20.”

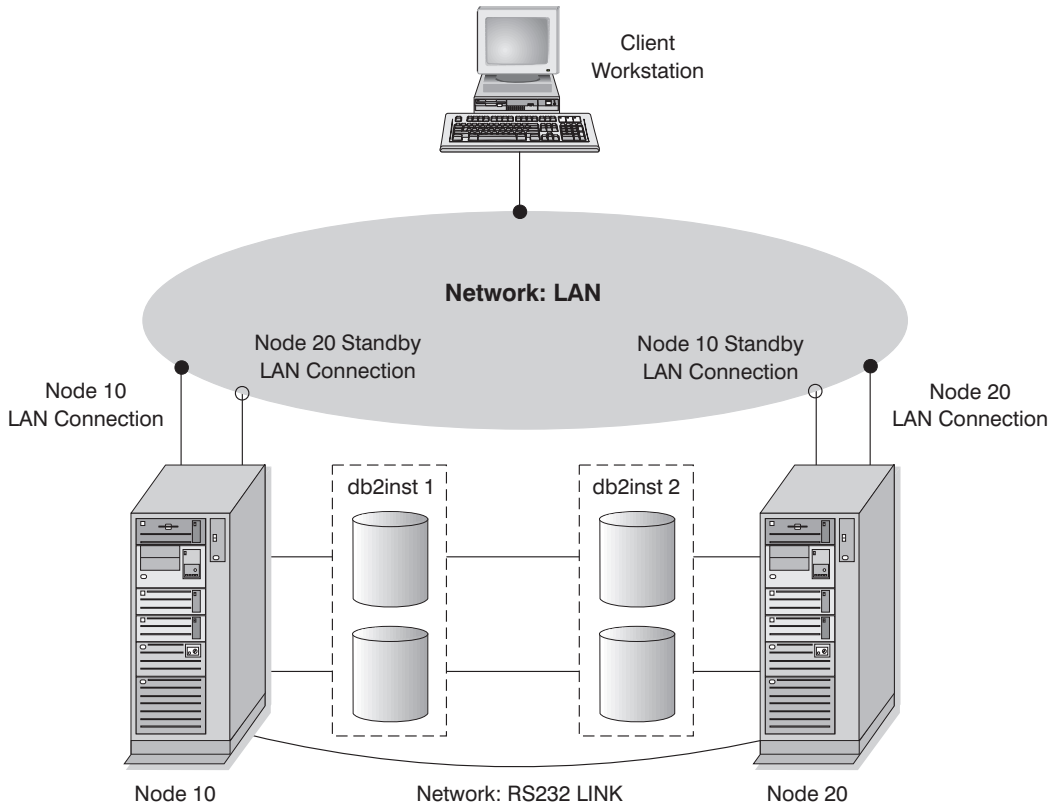


Figure 74. Instance Failover Example

In this example, we have two instances “db2inst1” and “db2inst2”: both are instances created from a single installation path on a shared filesystem. Instance “db2inst1” is created with a path of

```
/u/db2inst1
```

and instance “db2inst2” is created with a path of

```
/u/db2inst2
```

Both of these paths are on a shared filesystem accessible to both processors. Each instance has a single database, with a unique path, again on a shared resource accessible by both processors.

Both instances are accessed via remote clients over the TCP/IP protocol: “db2inst1” uses the service name “db2inst1_port” (port number 5500) and “db2inst2” uses the service name “db2inst2_port” (port number 5550). Remote clients accessing the “db2inst1” instance have this instance cataloged in their node directory using “node10” as the host name. Remote clients accessing the “db2inst2” instance have this instance cataloged in their node directory using “node20” as the host name. Under normal operating conditions, “db2inst1” is executing on “node10” and “db2inst2” is executing on “node20.” If “node10” were to fail, the failover script will start “db2inst1” on “node20”

and the external IP address associated with “node10” will be switched over to “node20.” Once the instance has been started by the failover script and the database restarted, the remote clients accessing this instance can connect to the database within this instance as if it were executing on “node10.”

Sample script:

hacmp-s4.sh

Mutual DB2 Partition Failover

Mutual failover of partitions in a partitioned database server environment requires that the failover of the partition occur as a logical node on the failover processor. If we have two partitions of a partitioned database server running on separate processors of a two processor HACMP cluster configured for mutual takeover, the partitions must failover as logical nodes. The default partition at each node must be defined as logical node 0, this means that when a partition fails over from one processor to another it will start as a logical node which does not have any direct remote communication protocol listeners. As such, the partition cannot be used as a coordinator node.

One other important consideration when configuring a system for mutual partition takeover concerns the local partition database path. When a database is created in a partitioned database environment, it is created on a root path which is not shared across the partitioned database servers. For example, consider the following statement:

```
CREATE DATABASE db_a1 ON /dbpath
```

This statement is executed under instance “db2inst” and creates the database `db_a1` on the path `/dbpath`. Each partition creates its actual database partition on its local `/dbpath` filesystem under `/dbpath/db2inst/nodexxxx` where `xxxx` represents the node number. With HACMP failover it will attempt to mount the `/dbpath` filesystem which is already being used by the other processor. As such, the failover script must mount the filesystem under a different logical point and set up a symbolic link from that filesystem to the appropriate `/dpath/db2inst/nodexxxx` path.

The following example shows a portion of the db2nodes.cfg file before and after the failover. In this example, node number 2 is running on processor 1 of the HACMP machine which has a hostname of “node201” and the netname is the same. Node number 3 is running on processor 2 of the HACMP machine which has a hostname of “node202” and again the netname is the same. The failover script will execute the command between the before and after definitions.

```
Before:
        1 node101 0 node101
        2 node201 0 node201    <= HACMP
        3 node202 0 node202    <= HACMP
        4 node301 0 node301

        db2start nodenum 2 restart node202 port 1 netname node202

After:
        1 node101 0 node101
        2 node202 1 node202    <= HACMP
        3 node202 0 node202    <= HACMP
        4 node301 0 node301
```

After the failover, any remote clients trying to directly access node number 2 as the coordinator will have to re-catalog the node entry for the database to point to the failover node. It is not recommended that you use a mutual failover scenario for coordinator nodes. If you require redundancy with your coordinator node, you should you use the hot standby mode.

Sample script:

hacmp-s5.sh

Additional HACMP Resources

For a complete understanding of the HACMP concepts, installation and configuration refer to the following books:

- *HACMP for AIX, Version 4.2: Concepts and Facilities*, SC23-1938
- *HACMP for AIX, Version 4.2: Installation Guide*, SC23-1940
- *HACMP for AIX, Version 4.2: Planning Guide*, SC23-1939

Appendix R. Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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 the

IBM Director of Licensing,
IBM Corporation,
500 Columbus Avenue,
Thornwood, NY, 10594
USA.

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

IBM Canada Limited
Department 071
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

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

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

ACF/VTAM
ADSTAR
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
CICS
C Set++
C/370
DATABASE 2
DatagLANce
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
Lan Distance
MVS/ESA
MVS/XA
NetView
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
SAA
SP
SQL/DS
SQL/400
S/370
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WIN-OS/2

Trademarks of Other Companies

The following terms are trademarks or registered trademarks of the companies listed:

C-bus is a trademark of Corollary, Inc.

HP-UX is a trademark of Hewlett-Packard.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Solaris is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

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

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Contacting IBM

This section lists ways you can get more information from IBM.

If you have a technical problem, please take the time to review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. Depending on the nature of your problem or concern, this guide will suggest information you can gather to help us to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

Telephone

If you live in the U.S.A., call one of the following numbers:

- 1-800-237-5511 to learn about available service options.
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, see Appendix A of the IBM Software Support Handbook. You can access this document by selecting the "Roadmap to IBM Support" item at: http://www.ibm.com/support/.

Note that in some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

World Wide Web
http://www.software.ibm.com/data/
http://www.software.ibm.com/data/db2/library/

The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more. The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information. (Note that this information may be in English only.)

Anonymous FTP Sites
ftp.software.ibm.com

Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools concerning DB2 and many related products.

Internet Newsgroups
comp.databases.ibm-db2, bit.listserv.db2-l

These newsgroups are available for users to discuss their experiences with DB2 products.

CompuServe
GO IBMDB2 to access the IBM DB2 Family forums

All DB2 products are supported through these forums.

To find out about the IBM Professional Certification Program for DB2 Universal Database, go to http://www.software.ibm.com/data/db2/db2tech/db2cert.html

Printed in the USA

S10J-8157-00