IBM DB2 Universal Database

Administration Guide: Design and Implementation

Version 6

SC09-2839-00

IBM DB2 Universal Database

Administration Guide: Design and Implementation

Version 6

SC09-2839-00

Before using this information and the product it supports, be sure to read the general information under “Appendix P. Notices” on page 861.

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

Contents

About This Book

The Administration Guide in its two volumes provides information necessary to use and administer the year 2000 ready, DB2* relational database management system (RDBMS) products, including:

- Information required for designing, implementing and managing databases (found in *Administration Guide, Design and Implementation*)
- Information regarding the configuring and tuning of your database environment to improve performance (found in *Administration Guide, 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 *Administrative 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.

  Error conditions when using the Control Center are recorded in the Control Center Administration Engine Log (db2cc.log). This log records information about the errors generated while using the Control Center. The log is always active while the Control Center is active. The log file is kept in the home directory of the executable that invokes the Control Center. That is, in the `bin` subdirectory of the `sqllib` subdirectory. The file can be viewed and updated using an ASCII file editor.

  The log file records the error message type, a time stamp, a process identifier (PID), a thread identifier (TID), and an SQL error message. The

process ID and the thread ID are used to identify the operating system that generated the log. Combined with the Control Center trace information, DB2 Service and Support personnel are able to determine which Control Center task caused the error. The information is only of use to the DB2 Service and Support personnel.

The log file can be edited by an ASCII file editor to remove log records that are no longer needed.

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, Design and Implementation* contains information about the following major topics:

Database Concepts

- Chapter 1. Introduction to Concepts Within DB2 Universal Database, presents an overview of DB2 Universal Database including: Using the Control Center, the types of parallelism provided by DB2, and federated systems use.

Database Design and Implementation

- Chapter 2. Designing Your Logical Database, discusses the concepts and guidelines for designing a logical database.

- Chapter 3. Designing Your Physical Database, discusses the guidelines for designing a physical database, including considerations related to physical data storage.
- Chapter 4. Implementing Your Design, discusses the concepts and guidelines for creating a database and the objects within a database.
- Chapter 5. Administering DB2 Using GUI Tools, introduces the Graphical User Interface (GUI) tools used to administer the database.
- Chapter 6. Controlling Database Access, describes how you can control access to your database's resources.
- Chapter 7. Auditing DB2 Activities, describes how you can detect and monitor unwanted or unanticipated access to data.
- Chapter 8. Utilities for Moving Data, discusses the LOAD, AutoLoader, IMPORT and EXPORT utilities. db2move and replication are also discussed.
- Chapter 9. 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 10. Distributed Databases, discusses how you can access multiple databases in a single transaction.
- Chapter 11. Using DB2 with an XA-Compliant Transaction Manager, discusses how you can use your databases in a distributed transaction processing environment such as CICS.

High Availability Systems

- Chapter 12. High Availability Cluster Multi-Processing (HACMP) on AIX, discusses the support of IBM High Availability Cluster Multi-Processing (HACMP) for AIX by DB2.
- Chapter 13. High Availability Cluster Multi-Processing, Enhanced Scalability (HACMP ES) for AIX, discusses the support of IBM High Availability Cluster Multi-Processing, Enhanced Scalability (HACMP ES) for AIX by DB2.
- Chapter 14. High Availability in the Windows NT Environment, discusses the support of Microsoft Cluster Server for Windows NT by DB2.
- Chapter 15. High Availability in the Solaris Operating Environment, Single-Partition Database, discusses the support of Sun Cluster 2.1 for the Sun Solaris Operating System by DB2.
- Chapter 16. High Availability in the Solaris Operating Environment, Partitioned Database, discusses the support of Sun Cluster 2.1 for the Sun Solaris Operating System by DB2 Enterprise - Extended Edition.

Appendixes

- Appendix A. How the DB2 Library Is Structured, provides information about the structure of the DB2 library, including SmartGuides, online help, messages, and books.
- Appendix B. Planning Database Migration, provides information about migrating databases to Version 5.
- Appendix C. Incompatibilities Between Releases, presents the incompatibilities introduced with Version 5.
- Appendix D. Naming Rules, provides the rules to follow when naming databases and objects.
- Appendix E. Using Distributed Computing Environment (DCE) Directory Services, provides information about how you can use DCE Directory Services.
- Appendix F. X/Open Distributed Transaction Processing Model, provides an overview of the X/Open Distributed Transaction Processing model and the DB2 database support provided.
- Appendix G. 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 H. National Language Support (NLS), introduces DB2 National Language Support (NLS) including information about countries, languages, and code pages.
- Appendix I. Issuing Commands to Multiple Database Partition Servers, discusses the use of the *db2_all* and *rah* shell scripts to send commands to all partitions in a partitioned database environment.
- Appendix J. How DB2 for Windows NT Works with Windows NT Security, describes how DB2 works with Windows NT security.
- Appendix L. Configuring Multiple Logical Nodes, describes how to configure multiple logical nodes in a partitioned database environment.
- Appendix M. Using Virtual Interface (VI) Architecture, describes how to enable Virtual Interface Architecture for use with DB2 Enterprise - Extended Edition in the Windows NT environment.
- Appendix N. Lightweight Directory Access Protocol (LDAP) Directory Services, provides information about how you can use Lightweight Directory Access Protocol (LDAP) Directory Services.
- Appendix O. Extending the Control Center, provides information about how you can extend the Control Center by adding new tool bar buttons including new actions, adding new object definitions, and adding new action definitions.

The other volume of the Administration Guide (*Administration Guide, Performance*) is concerned with performance issues. That is, those topics and issues concerned with establishing, testing, and improving all aspects of your application's, and the DB2 Universal Database product, performance.

The specific chapters and appendixes in that volume are briefly described here:

Introduction to Performance

- Elements of Performance, introduces concepts and considerations for managing and improving DB2 UDB performance.

Tuning Application Performance

- Application Considerations, describes some techniques for improving database performance when designing your applications.
- Environmental Considerations, describes some techniques for improving database performance when setting up your database environment.
- System Catalog Statistics, describes how statistics about your data can be collected and used to ensure optimal performance.
- Understanding the SQL Compiler, describes what happens to an SQL statement when it is compiled using the SQL compiler.
- 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

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

Appendixes

- DB2 Registry and Environment Variables, presents profile registry values and environment variables.
- Sample Tables, contains a description of the sample tables provided with the database manager.
- Catalog Views, contains a description of each system catalog view, including column names and data types.

- Explain Tables and Definitions, provides information about the tables used by the DB2 Explain facility and how to create those tables.
- SQL Explain Tools, provides information on using the DB2 explain tools: db2expln and dynexpln.
- db2exfmt — Explain Table Format Tool, provides information on using the DB2 explain tool to format the explain table data.

Part 1. Database Concepts

Chapter 1. Introduction to Concepts Within 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
- Overview of federated database system concepts
- Enabling federated database systems

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 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 *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 6 for more information about nodegroups. See “Overview of DB2 Parallelism Concepts” on page 5 for the definition of a database partition.

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 *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 query is used in applications or by users to retrieve data from a database. The query uses Structured Query Language (SQL) to create a statement in the form of

```
SELECT <data_name> FROM <table_name>
```

In this chapter we use the term “query” to identify a retrieval request (a SELECT statement) from a database.

Figure 1 on page 5 illustrates the relationship among the objects just described. It also illustrates that tables, indexes, and long data are stored in table spaces.

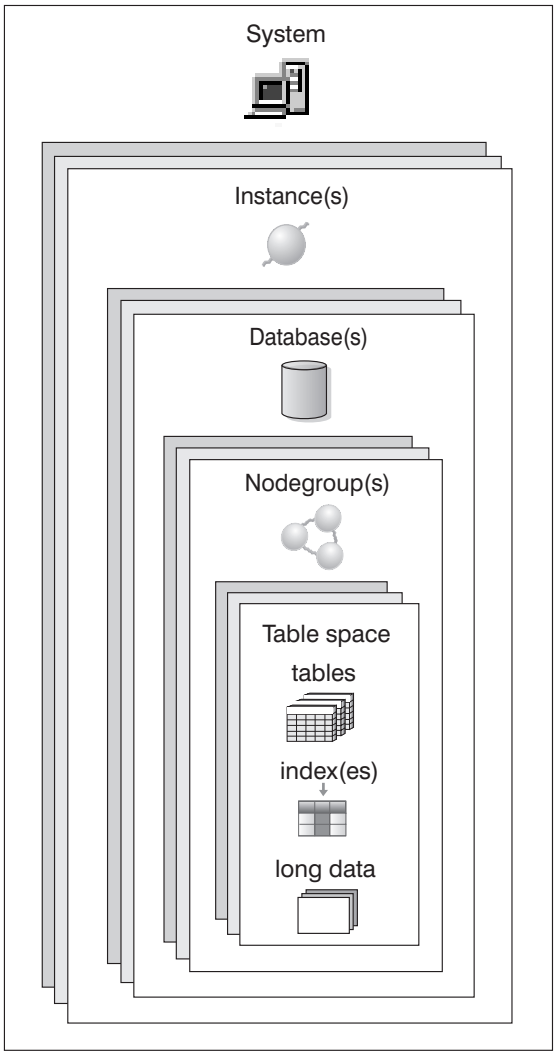


Figure 1. Relationship Among Some Database Objects

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. 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 7 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 two database partitions.
- The database partition within Nodegroup 2 is shared (and overlaps) with Nodegroup 1.
- There is a single database partition within Nodegroup 3 that is shared (and overlaps) with Nodegroup 1.

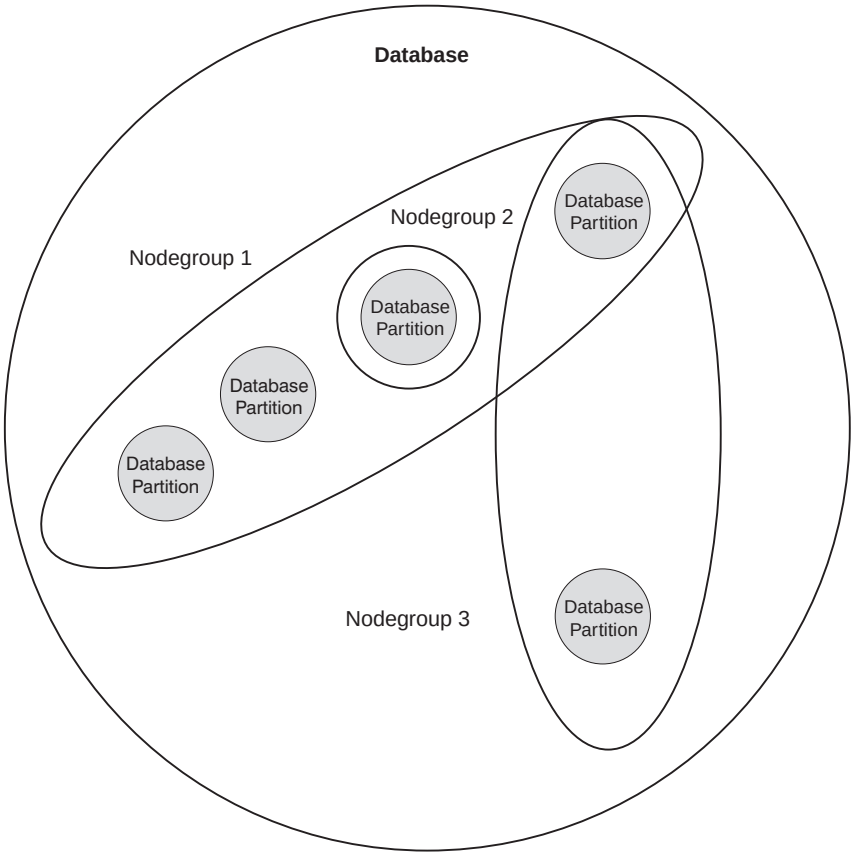


Figure 2. Nodegroups in a Database

You create a new nodegroup using the CREATE NODEGROUP statement. Refer to 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 67.

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 12. Table 1 on page 20 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 11.)

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.*

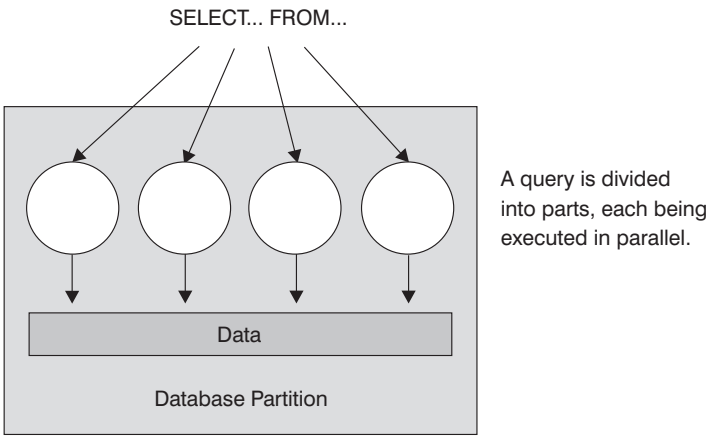


Figure 3. Intra-Partition Parallelism

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 20 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 11.)

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.*

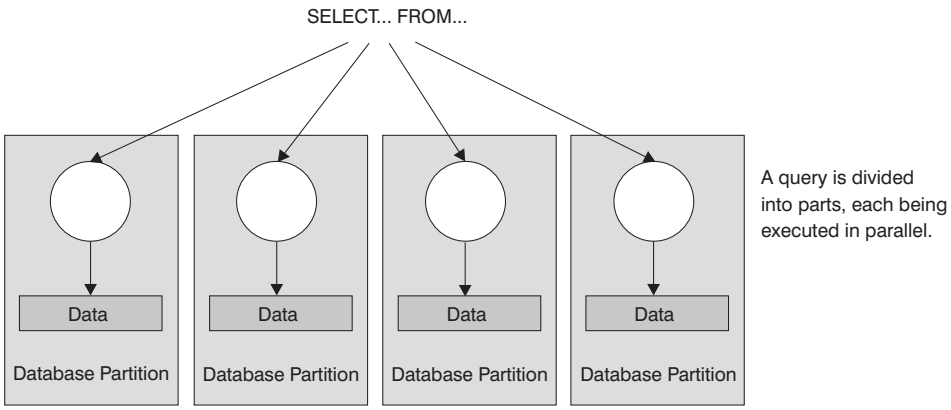


Figure 4. Inter-Partition Parallelism

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 20 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 11 illustrates this.

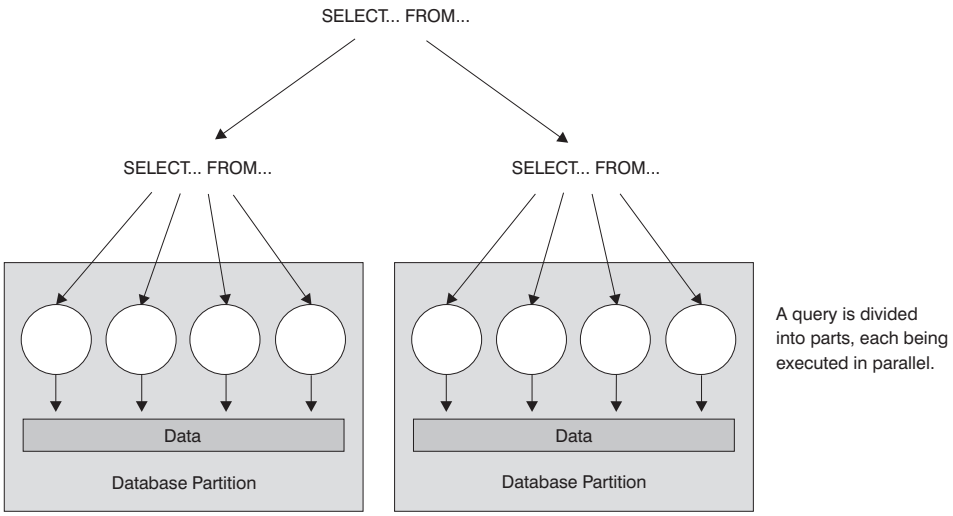


Figure 5. Both Inter-Partition and Intra-Partition Parallelism

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. Refer to the *Data Movement Utilities Guide and Reference* or the LOAD command in the *Command Reference* for information on how to enable parallelism for the LOAD utility.

In a partitioned database environment, the AutoLoader utility takes advantage of intra-partition, inter-partition, and I/O parallelism by parallel invocations of load at each database partition where the table resides. Refer to *Data Movement Utilities Guide and Reference* for more information about the AutoLoader 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. Refer to 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 13 illustrates this environment.

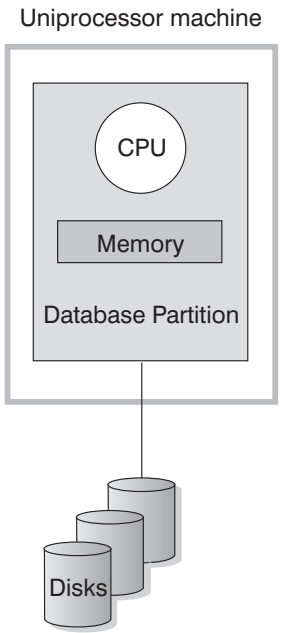


Figure 6. Single Partition On a Single Processor

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 20 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.

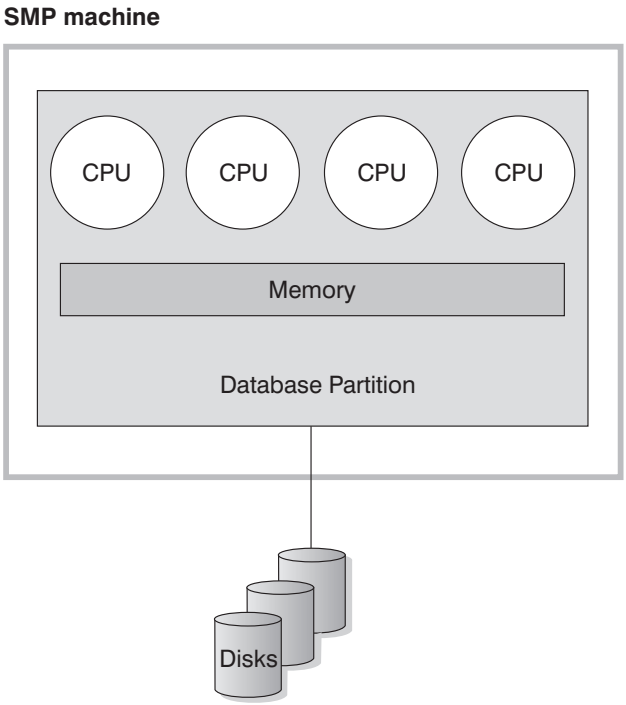


Figure 7. Single Partition Database Symmetric Multiprocessor System

Table 1 on page 20 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 16 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.

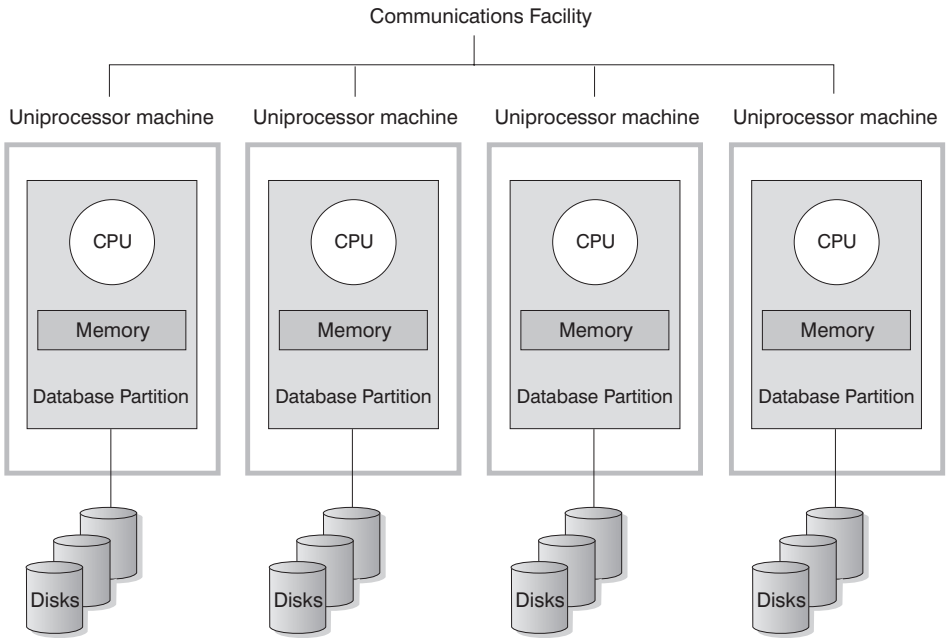


Figure 8. Massively Parallel Processing System

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 20 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.

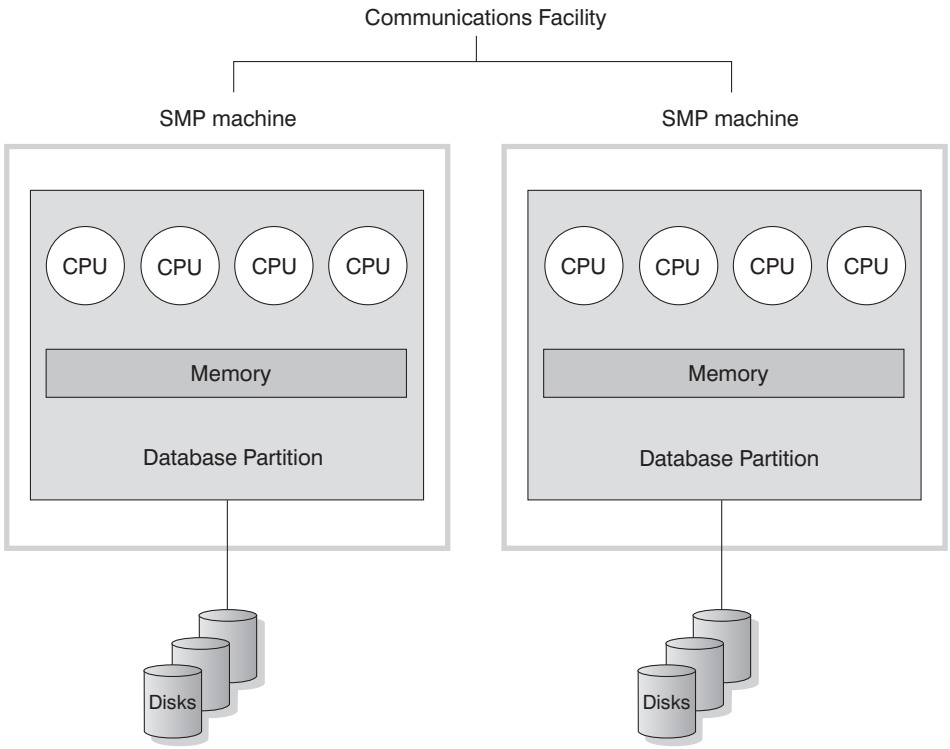


Figure 9. Cluster of SMPs

Table 1 on page 20 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 processors are manufactured. Figure 10 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.

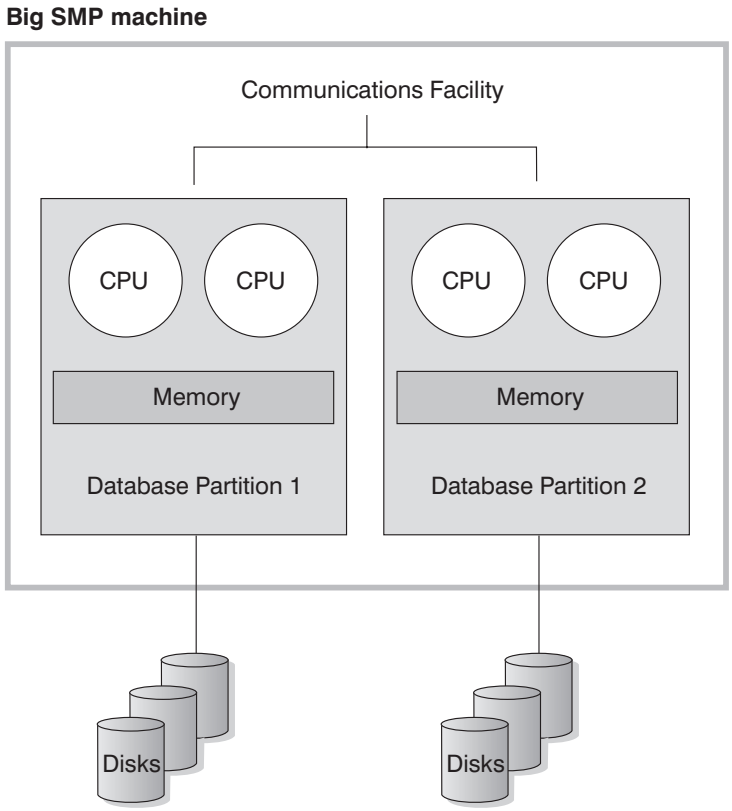


Figure 10. Partitioned Database, Symmetric Multiprocessor System

Figure 11 illustrates the fact that you can multiply the configuration shown in Figure 10 on page 18 to increase processing power.

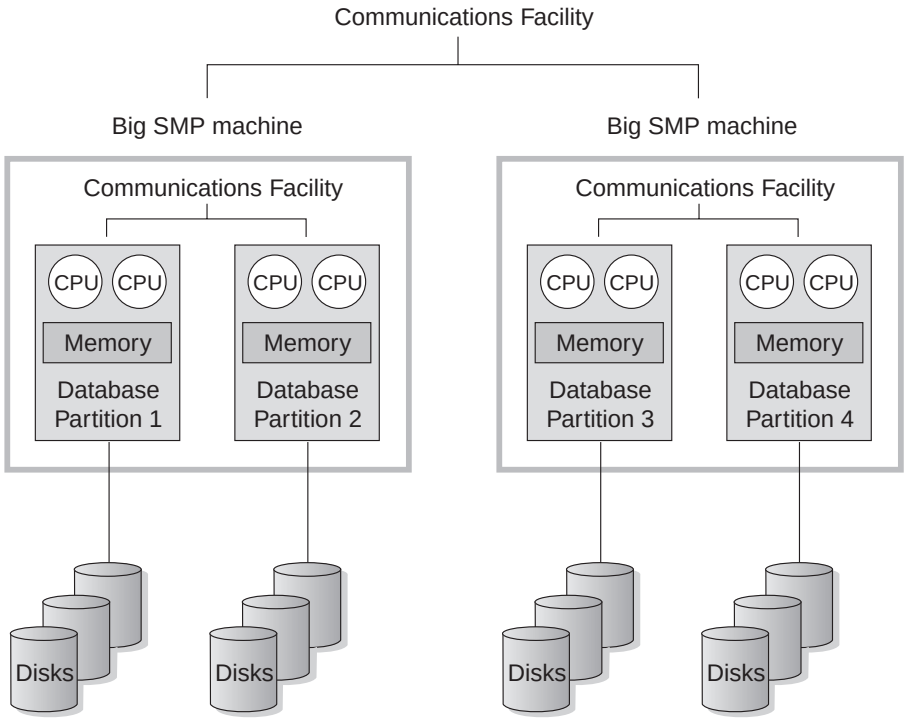


Figure 11. Partitioned Database, Symmetric Multiprocessor Systems Clustered Together

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 failover strategies. See "Chapter 12. High Availability Cluster Multi-Processing (HACMP) on AIX" on page 515 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 20 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
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. Refer to *Administration Guide, Performance* for more information on this parameter.

DFT_DEGREE
Database configuration parameter. Provides the default for the DEGREE bind option and the CURRENT DEGREE special register. Refer to *Administration Guide, Performance* for more information on this parameter.

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

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

For more information on the configuration parameter settings, and how to enable applications to process in parallel, refer to ″Configuring DB2″ in the *Administration Guide, Performance.*

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

The Load utility automatically makes use of parallelism, or you can use the following parameters on the LOAD command:

- CPU_PARALLELISM
- DISK_PARALLELISM

Refer to the *Data Movement Utilities Guide and Reference* for information on the LOAD command.

AutoLoader

You can enable multiple split processes for the AutoLoader by specifying the MODIFIED BY ANYORDER parameter for the LOAD specification in the *autoloader.cfg* file. Refer to *Administration Guide, Performance* for more information on this LOAD specification and the configuration file.

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
- Multiple processors must be enabled on an SMP machine.

Refer to the *SQL Reference* for information on the CREATE INDEX statement.

Backup Database / Table Space

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.
- Use the PARALLELISM parameter on the BACKUP command to specify the degree of parallelism.
- Use the WITH `num-buffers` BUFFERS parameter on the BACKUP command to ensure enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

  Also, use a backup buffer size that is:

  – As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.

  – At least as large as the largest (extentsize * number of containers) product of the table spaces being backed up.

Refer to the *Command Reference* for information on the BACKUP DATABASE command.

Restore Database / Table Space

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.
- Use the PARALLELISM parameter on the RESTORE command to specify the degree of parallelism.
- Use the WITH `num-buffers` BUFFERS parameter on the RESTORE command to ensure enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

  Also, use a restore buffer size that is:

  – As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.

- At least as large as the largest (extentsize * number of containers) product of the table spaces being restored.
- The same as, or an even multiple of, the backup buffer size.

Refer to the *Command Reference* for information on the RESTORE DATABASE command.

Federated Database System Concepts

A *federated database system* or *federated system* is a database management system (DBMS) that supports applications and users submitting SQL statements referencing two or more DBMSs or databases in a single statement. An example is a join between tables in two different DB2 databases. This type of statement is called a *distributed request*.

A DB2 UDB Version 6 federated system provides support for distributed requests across databases and DBMSs. You can, for example, perform a UNION operation between a DB2 table and an Oracle view. Supported DBMSs include DB2, members of the DB2 Family (such as DB2 for OS/390 and DB2 for AS/400), and Oracle.

A DB2 federated system provides *location transparency* for database objects. If information (tables and views) is moved, references to that information (called *nicknames*) can be updated without any changes to applications that request the information. A DB2 federated system also provides *compensation* for DBMSs that do not support all of the DB2 SQL dialect or certain optimization capabilities. Operations that cannot be performed at a DBMS, such as recursive SQL, are run at DB2.

A DB2 federated system functions in a *semi-autonomous* manner: DB2 queries containing references to Oracle objects can be submitted while Oracle applications are accessing the same server. A DB2 federated system does not monopolize or restrict access to Oracle or other DBMS objects (beyond integrity/locking constraints).

A DB2 federated system consists of a DB2 UDB Version 6 instance, a database that will serve as the *federated database*, and one or more *data sources*. The federated database contains catalog entries identifying data sources and their characteristics. A data source consists of a DBMS and data. Applications connect to the federated database just like any other DB2 database. See Figure 12 on page 24 for a visual representation of a federated database environment.

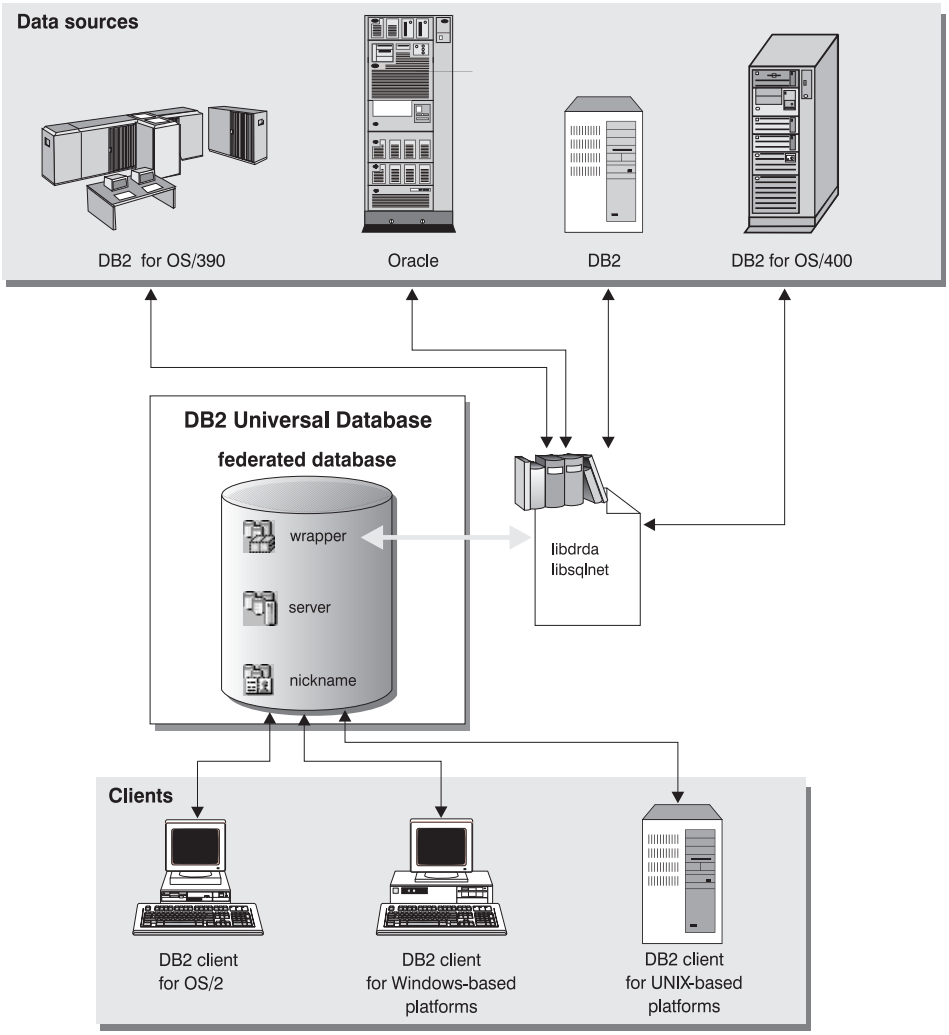


Figure 12. A Federated Database System

DB2 federated database catalog entries contain information about data source objects: what they are called, information they contain, and conditions under which they can be used. Because this DB2 catalog stores information about objects in many DBMSs, it is called a *global catalog*. Object attributes are stored in the catalog. The actual DBMSs being referenced, modules used to communicate with the data source, and DBMS data objects (such as tables) that will be accessed are outside the database. (One exception: the federated database can be a data source for the federated system.) You can create federated objects using the Control Center or SQL DDL statements. Required federated database objects are:

Wrappers
Identify the modules (dll, library, etc.) used to access a particular class or category of data source.

Servers
Define data sources. Server data includes the wrapper name, server name, server type, server version, authorization information, and server options.

Nicknames
Identifiers stored in the federated database that reference specific data source objects (tables, aliases, views). Applications reference nicknames in queries just like they reference tables and views.

Depending on your specific needs, you can create additional objects:

- User mappings, to address authentication issues
- Data type mappings, to customize the relationship between a data source type and an DB2 type
- Function mappings, to map a local function to a data source function
- Index specifications, to speed performance

After a federated system is set up, the information in data sources can be accessed as if it was in one big database. Users and applications send queries to one federated database, which then retrieves data from DB2 Family and Oracle systems as needed. User and applications specify nicknames in queries; these nicknames provide references to tables and views located at data sources. From an end-user perspective, nicknames are similar to aliases.

There are many factors affecting federated system performance. The most critical step is to ensure that accurate and up-to-date information about data sources and their objects is stored in the federated database global catalog. This information is used by the DB2 optimizer and can affect decisions to push down operations for evaluation at data sources. See the *Administration Guide, Performance* for additional information on federated system performance.

A DB2 federated system operates under some restrictions. Distributed requests are limited to read-only operations. In addition, you cannot execute utility operations (LOAD, REORG, REORGCHK, IMPORT, RUNSTATS, and so on) against nicknames.

You can, however, use a pass-through facility to submit DDL and DML statements directly to database managers using the SQL dialect associated with that data source.

Federated systems tolerate parallel environments. Performance gains are delimited by the extent to which a federated database query can be semantically broken down into local object (table, view) references and nickname references. Requests for nickname data are processed sequentially; local objects can be processed in parallel. For example, given the query `SELECT * FROM A, B, C, D` where A and B are local tables and C and D are nicknames referencing tables at Oracle data sources, one possible plan would join tables A and B with a parallel join. The results are then joined sequentially with nicknames C and D.

Enabling a Federated System

DB2 Extended Edition and DB2 Enterprise Extended Edition can support federated databases. To enable a federated system:

1. Select the *distributed join* installation option of DB2 EE or EEE during installation
2. Set the database manager configuration parameter *federated* to "YES"
3. Create wrappers, servers, and nicknames (see "Creating a Database" on page 145 for more information)
4. Create additional objects or set options as required (see "Chapter 4. Implementing Your Design" on page 99 for more information)

Part 2. Database Design and Implementation

Chapter 2. 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 31
- "Provide Column Definitions for All Tables" on page 34
- "Identify One or More Columns as a Primary Key" on page 36
- "Be Sure Equal Values Represent the Same Entity" on page 38
- "Consider Normalizing Your Tables" on page 39
- "Planning for Constraint Enforcement" on page 44
- "Other Database Design Considerations" on page 51.

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 3. Designing Your Physical Database" on page 55 and "Chapter 4. Implementing Your Design" on page 99.

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. (Refer to the "Sample Tables" in the *Administration Guide, Performance,* 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

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 13. 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 14. 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 15. 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 691.

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, refer to 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 *structured type* is a user-defined type that has a structure that is defined in the database. It contains a sequence of named *attributes*, each of which has a data type. A structured type may be defined as a *subtype* of another structured type, called its *supertype*. A subtype inherits all the attributes of its supertype and may have additional attributes defined. The set of structured types that are related to a common supertype is called a *type hierarchy* and the supertype that does not have any supertype is called the *root type* of the type hierarchy.

A structured type may be used as the type of a table or a view. The names and data types of the attributes of the structured types, together with the object identifier, become the names and data types of the columns of this *typed table* or *typed view*. Rows of the typed table or typed view can be thought of as a representation of instances of the structured type.

A structured type cannot be used as the data type of a column of a table or a view. There is also no support for retrieving a whole structured type instance into a host variable in an application program.

A *reference type* is a companion type to the structured type. Similar to a distinct type, a reference type is a scalar type that shares a common representation with one of the built-in data types. This same representation is shared for all types in the type hierarchy. The reference type representation is defined when the root type of a type hierarchy is created. When using a reference type, a structured type is specified as a parameter of the type. This parameter is called the *target type* of the reference.

The target of a reference is always a row in a typed table or view. When a reference type is used, it may have a *scope* defined. The scope identifies a table (called the *target table*) or view (called the *target view*) that contains the target row of a reference value. The target table or view must have the same type as the target type of the reference type. An instance of a scoped reference type uniquely identifies a row in a typed table or typed view, called its *target row*.

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, structured types, reference 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 46, 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 16. 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 40)

Second Each column that is not in the key provides a fact that depends on the entire key. (See “Second Normal Form” on page 40)

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 42)

Fourth No row contains two or more independent multi-valued facts about an entity. (See “Fourth Normal Form” on page 43)

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

Table 8. Table Violates Second Normal Form (continued)

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY	WAREHOUSE_ADDRESS
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 on page 40, 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, refer to "Tuning Application Performance" in the *Administration Guide, Performance.*

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 and to allow the optimizer to exploit knowledge of these special relationships to process queries more effectively. 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. Refer to 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. Refer to 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.

Refer to the *Data Movement Utilities Guide and Reference* 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 174 and refer to the *Application Development 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*, *data typing*, 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. Another way to carry out audit activities is through the use of the DB2 audit facility. See "Chapter 7. Auditing DB2 Activities" on page 333 for more information.

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 have a table that has all of your employee information in it. However, you would like to have the employee information divided into separate tables by division or department. In this case, a summary table for each division or department based on the data in the original table would be helpful. See "Creating a Summary Table" on page 187 for more information on summary tables.

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 6. Controlling Database Access" on page 281.

You can create tables that have a *structured type* associated with them. With such typed tables, you can establish a hierarchical structure with a defined relationship between those tables called a *type hierarchy*. The type hierarchy is made up of a single root type, supertypes, and subtypes.

A *reference type* representation is defined when the root type of a type hierarchy is created. The target of a reference is always a row in a typed table or view.

See "Chapter 4. Implementing Your Design" on page 99 for more information on implementing a design that includes typed rows and tables. Refer to *Data Movement Utilities Guide and Reference* for more information on moving data between typed tables that are in a hierarchical structure.

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 3. Designing Your Physical Database

After you have completed Chapter 2. Designing Your Logical Database and before Chapter 4. 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
- Federated Database Design Considerations

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 102 for an introduction to instances.

The naming scheme used on UNIX platforms is

```
specified_path/$DB2INSTANCE/NODEnnnn/SQL00001
```

The naming scheme used on Intel platforms is

```
D:\$DB2INSTANCE\NODEnnnn\SQL00001
```

where

- `specified_path` is the optional, user-specified location to install the instance.

- NODEnnnn is the node identifier in a partitioned database environment. The first node is NODE0000.
- "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. Refer to *Administration Guide, Performance* 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.

DB2RHIST.ASC
This file is the database history file. It keeps a history of administrative operations on the database, such as when performing backups and restoring a backup.

DB2RHIST.BAK
This file is a backup copy of DB2RHIST.ASC.

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" on page 59
- "User Table Data" on page 59
- "Long Field Data" on page 61
- "Large Object (LOB) Data" on page 62
- "Index Space" on page 63

After reading these sections, you should read "Designing and Choosing Table Spaces" on page 75.

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.

The are several other means available to you to assist you in determining the size of various parts of your database.

From the Control Center, you could select an object and choose to use the "Estimate Size" utility. The utility can be used to tell you the current size of an existing object like a table. You can then change certain aspects of the object, and the utility will calculate the new, estimated values for the object. The utility will give you a way of determining the approximate DASD requirements for current and future growth estimation. The utility gives more than a single estimate of the size of the object, by also providing two size extremes possible for the object: both the smallest size based on the given values as well as the largest possible size.

From the Control Center, you can determine the relationships between objects by using the “Show Related” dialog.

From the Control Center, you can select any database object on the instance and request “Generate DDL”. This function uses the *db2look* utility to generate data definition statements for the database.

In all of these cases, either the “Show SQL” or “Show Command” buttons are made available to you. You can also save the resulting SQL or commands as script files to be used again at another time. And all of the dialogs and utilities have online help to assist you with your using these facilities.

You should keep these facilities in mind as you work through your planning of your physical database requirements.

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 2.5 MB 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 20 MB of space. For more information, see “Designing and Choosing Table Spaces” on page 75.

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

By default, table data is stored on 4 KB pages. Each 4 KB 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. You can have a maximum of 500 columns when using a 4 KB page size.

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” on page 61 and “Large Object (LOB) Data” on page 62 for information about estimating the space required for the table objects that will contain the data stored using these data types.)

Typically, rows are inserted into the table in an approximate 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 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.

If the ALTER TABLE APPEND ON statement is used, then data will always be appended and information about any free space on the data pages will not be kept.

See "Long Field Data" on page 61 and "Large Object (LOB) Data" on page 62 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 4 KB pages can be estimated by calculating:

```
ROUND DOWN(4020/(average row size + 10)) = records_per_page
```

Then use records_per_page with:

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

Note: This formula is only an estimate and is not guaranteed to be accurate. Accuracy of the estimate lessens if the length of records varies due to fragmentation and overflow records.

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

The factor of "1.1" is for overhead.

You also have the option to create buffer pools or table spaces that have an 8 KB, 16 KB, or 32 KB page size. All tables created within a table space of a particular size will have a matching page size. A single table or index object can then be as large as 512 GB in size (based on a 32 KB page size; the maximum varies dependent on the page size). You can have a maximum of 1012 columns when using an 8 KB, 16 KB, or 32 KB page size. The maximum number of columns is 500 for a 4 KB page size. Row lengths vary based on page size:

- When the page size is 4 KB, the row length can be up to 4005 bytes in length.
- When the page size is 8 KB, the row length can be up to 8101 bytes in length.

- When the page size is 16 KB, the row length can be up to 16 293 bytes in length.
- When the page size is 32 KB, the row length can be up to 32 677 bytes in length.

Having a larger page size allows for the possible reduction in the number of levels in any index. If you are working with OLTP applications which do random row reads and writes, a smaller page size is better because it wastes less buffer space with undesired rows. If you are working with DSS applications which access large numbers of consecutive rows at a time, a larger page size is better because it reduces the number of Input/Output requests required to read a specific number of rows. An exception in this latter cases occurs when the row size is smaller than the page size divided by 255. In such a case, there is wasted space on each page. (Recall that there can only be a maximum of 255 rows per page.) To reduce this wasted space, a smaller page size may be more appropriate.

There are some restrictions when using a page size larger than 4 KB. When conducting backup and restore operations, you cannot restore a backup to a different page size.

No matter the page size, you cannot import IXF data files that represent more than 755 columns.

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” on page 59 and “Large Object (LOB) Data” on page 62).

Data is stored in 32 KB 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 4 KB 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 the page size, 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 59 and "Long Field Data" on page 61).

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 64 MB 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 64 MB.)

  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 1 KB boundary. Without the COMPACT option, there is no attempt to reduce the space used to the nearest 1 KB 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 4 KB allocation pages separated from the actual data. The number of these 4 KB pages is dependent on the amount of data, including unused space, allocated for the large object data. The overhead is calculated as follows: one 4 KB pages for every 64 GB plus one 4 KB page for every 8 MB.

If character data is less than the page size, 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. Refer to 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 12 KB. For DMS, the minimum is an extent.

Following are two methods that you can use when calculating index space. The first method is a rough estimate, while the second method provides a more accurate estimate:

- The average number of keys per leaf page is roughly:

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

where:

– U = the usable space on a page is approximately equal to the page size minus 100. For a page size of 4096, U is 3996.

– M = U / (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 and an extra byte for the length of each variable length key part.

If there are include columns, they should be accounted for in minimumKeySize and averageKeySize.

The .9 can be replaced by any (100 - pctfree)/100, if a percent free other than the default of ten (10) percent was specified during the index creation.

- If you want a more accurate estimate:

```
L = number of leaf pages = X / (avg number of keys on leaf page)
```

Where X is the total number of rows in the table.

You can estimate the original size of an index as:

```
(L + 2L/(average number of keys on leaf page)) * pagesize
```

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.

You should provide additional space for index growth due to INSERT/UPDATE activity, which may result in page splits.

Use the following calculations to obtain a more accurate estimate of the original index size, as well as an estimate of the number of levels in the index. (This may be of particular interest if include columns are being used in the index definition.) The average number of keys per non leaf page is roughly:

```
   (.9 * (U - (M*2))) * (D + 1)
-------------------------------
      K + 12 + (8 * D)
```

Where:

U Is the same as for the leaf page calculation above.

D Is the average number of duplicates per key value on non leaf pages (this will be much smaller than on leaf pages and you may want to simplify by setting to 0).

M Is `U / (8 + minimumKeySize for non leaf pages)`.

K Is the `averageKeySize` for non leaf pages.

The `minimumKeySize` and `averageKeySize` will be the same as on leaf pages, except when there are include columns. Include columns are not stored on the non leaf pages, so the size of include columns should be excluded from the `minimumKeySize` and `averageKeySize` for non leaf page calculations.

You should not replace `.9` with `(100 - pctfree)/100` unless this value is greater than `.9`, because a maximum of 10% free space will be left on non-leaf pages during index creation.

The number of non-leaf pages can be estimated as follows:

P Is the number of pages (0 initially).

L Is the number of leaf pages.

N Is the number of keys for each non-leaf page.

Y Is `L/N`.

Z Is the number of levels in the tree (1 initially).

```
if L > 1 then {P++; Z++}
While (Y > 1)
{
   P = P + Y
   Y = Y / N
  Z++
}
```

So the total number of pages is `T = (L + P + 2) * 1.0002`. The additional .02% is for overhead such as space map pages.

The amount of space required to create the index is estimated as `T * pagesize`, and the number of levels in index tree is estimated to be `Z`.

Additional Space Requirements

Additional space is also required as follows:

- "Log File Space" on page 66
- "Temporary Work Space" on page 67

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
- *logsecond* is the number of secondary log files as defined in the database configuration file
- *logfilsiz* is the number of pages in each log file as defined in the database configuration file
- 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.

Refer to the *Administration Guide, Performance* for more information on the configuration parameters mentioned above.

Note: The total active log space cannot exceed 4 GB in size. That is, the previous calculation result cannot exceed 4 GB in size.

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 online 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:
  - online 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.
- Each database partition must be assigned a unique partition number. 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” on page 69
- “Partitioning Maps” on page 69
- “Partitioning Keys” on page 71

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.

An alternative to consider when you would like tables to be positioned on each database partition, is to use summary tables and then replicate those tables. A summary table could be created with the information you choose. Then you could replicate the summary table to each node. See "Replicated Summary Tables" on page 74 for more information on why you would want to do this.

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 4 096 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 17 shows how the row with the partitioning key value (c1, c2, c3) is mapped to partition 2, which, in turn, references database partition n5.

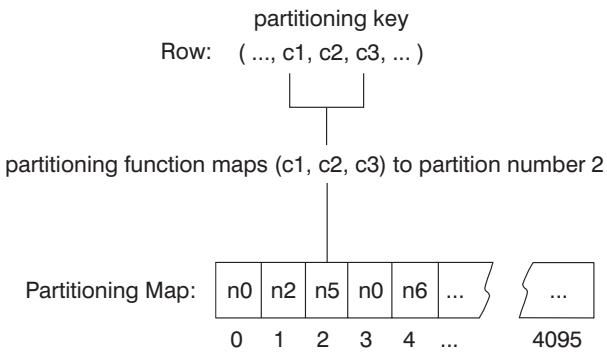


Figure 17. Data Distribution Using a Partition Map

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, refer to "Redistributing Data Across Database Partitions" in *Administration Guide, Performance*.

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, refer to the *Administrative 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 73. For more information on table collocation, see "Table Collocation" on page 73.

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 69.
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. Placing both tables in the same nodegroup ensures a common partitioning 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 partitioning key must be *partition-compatible*. For information about partition compatibility, see "Partition Compatibility".

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. Refer to "Collocated Joins" in the *Administration Guide, Performance* for more information.

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.
- BIGINT, 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.

Replicated Summary Tables

A *summary table* is a table that is defined by a query that is also used to determine the data in the table. Summary tables can be used to improve the performance of queries. If the database manager determines that a portion of a query could be resolved using a summary table, the query may be rewritten by the database manager to use the summary table. This decision is based on certain settings such as CURRENT REFRESH AGE and CURRENT QUERY OPTIMIZATION special registers.

In a partitioned database environment, you can replicate summary tables. You can use *replicated summary tables* to improve query performance. A replicated summary table is a table that is based on a table that you created in a table space (perhaps a table space created in a single-partition nodegroup), but you want all the table data replicated across all the database partitions in the nodegroup. To create the replicated summary table, you use the CREATE TABLE statement with the REPLICATED keyword. The REPLICATED keyword is only valid when the AS *fullselect* and REFRESH IMMEDIATE keywords are also used.

See “Creating a Summary Table” on page 187 for information concerning summary tables.

By using replicated summary tables, you can obtain collocation between tables that are not typically collocated. Replicated summary tables are particularly useful for joins in which you have a large fact table and small dimension tables. To minimize the extra storage required and the impact of having to update every replica, good candidates for tables to be replicated would have the following characteristics:

- They are small.
- They are infrequently updated.

Note: You should also consider replicating larger tables that are infrequently updated: in this situation, the one-time cost of replication is offset by the performance benefits that can be obtained by collocation.

By specifying a suitable predicate in the subselect used to define the replicated table, you can replicate both selected columns, selected rows, or both.

For more information about replicated summary tables, refer to the CREATE TABLE statement in the *SQL Reference*. Refer to “Collocated Joins” in the *Administration Guide, Performance* for more information.

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.

See “Creating a Table Space” on page 153 or “Altering a Table Space” on page 214 for information on how to create or alter a table space.

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.

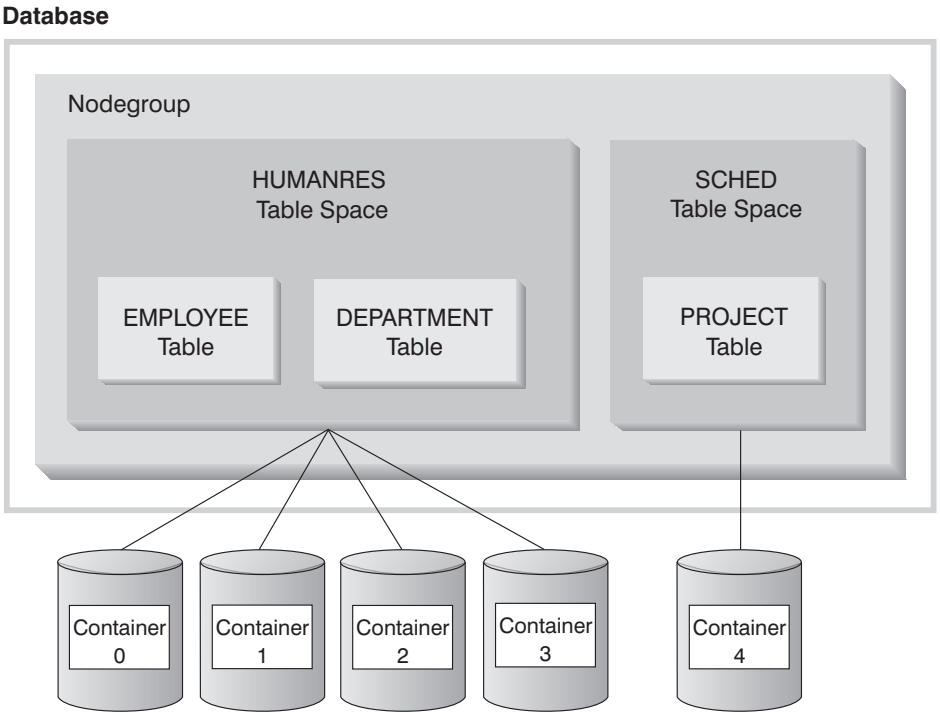


Figure 18. Table Spaces and Tables Within a Database

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 4 KB 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.

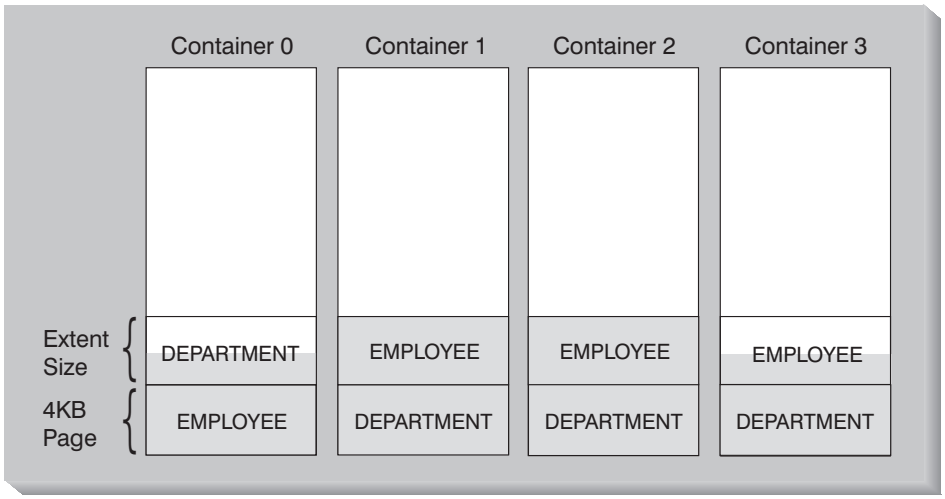


Figure 19. Use of Container and Extents

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 with a sufficient page size, then use it. Otherwise, if user-created table spaces exist, then choose one which is of the smallest page size that is sufficient for this table and use it. Otherwise, use USERSPACE1 if it exists with a sufficient page size. If none of these exist with a sufficient page size, then the table creation fails.

  The sufficient page size of a table is determined by either the byte count of the rows or the number of columns. The maximum number of bytes allowed in a row of a table is dependent on the page size of the table space in which the table is created. The possible values for the page size are 4 KB (the default), 8 KB, 16 KB, and 32 KB. You can use a table space with one page size for the base table, and a different table space with a different page size for LONG or LOB data. (Recall that SMS does not support tables that span table spaces, while DMS does.) If the number of columns or the row size exceeds the limits for a table space's page size, an error is returned (SQLSTATE 42997).

- 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.

  Note: If queries are executing against tables in table spaces that are defined with a page size of larger than the default 4 KB, some of them may fail because of the lack of a temporary table space defined with a larger page size (for example, an ORDER BY on 1012 columns). You may need to create a temporary table space with a larger page size (8 KB, 16 KB, or 32 KB). In fact, any Data Manipulation Language (DML) statement could fail unless there exists a temporary table space with the same page size as the largest page size of user data.

  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.

  You should define a single SMS temporary table space with a page size equal to the page size used in the majority of your regular table spaces. This should be suitable for typical environments and workloads. For detailed guidelines for those environment and workloads not as typical see "Recommendations for Temporary Table Spaces" on page 90.

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 85.

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 81 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. Refer to 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 limited 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.

Note: Care must be taken when defining the containers. There must not be any files or directories on the containers. If there are existing files or directories on the containers, error message "SQL0298N Bad container path." is reported.

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 89 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 (refer to the *Administration Guide, Performance* for more information on this parameter). 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 2 GB limit. Therefore, if you want a 64 GB 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: 64 GB (for 4 KB pages); 128 GB (for 8 KB pages); 256 GB (for 16 KB pages); 512 GB (for 32 KB pages)
  - Long field data: 2 TB
  - Temp data: 2 TB
- 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.
- Your workload involves LOBs or LONG VARCHARs and can benefit from file system caching.

  Note: LOBs and LONG VARCHARs are not buffered by DB2’s buffer pool.
- Some operating systems allow you to have physical devices greater than 2 GB 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 Temporary Table Spaces
- Recommendations for Catalog Table Spaces
- Workload Considerations
- Choosing an SMS or DMS Table Space
- Optimizing Performance When Data is Placed on RAID Devices.

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 4 KB 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, refer to the *Administration Guide, Performance*.

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. Refer to “Managing the Database Buffer Pool” in the *Administration Guide, Performance* for more information on this topic.

Note: If you have determined that a page size of 8 KB, 16 KB, or 32 KB is required within your database, then each table space with one of these page sizes must be mapped to a buffer pool with the same page size.

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 default buffer pools (one each of 4 KB, 8 KB, 16 KB, and 32 KB page sizes) 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, refer to 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, refer to the *SQL Reference* manual. For more information on nodegroups and physical database design, see the “Designing Nodegroups” on page 67.

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 89 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. (Refer to *Administration Guide, Performance* 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 Temporary Table Spaces

It is recommended that you define a single SMS temporary table space with a page size equal to the page size used in the majority of your regular table

spaces. This should be suitable for typical environments and workloads. However, it can be advantageous, in specific workloads, to experiment with different temporary table space configurations. The following points should be considered:

- Temporary tables are in most cases accessed in batches and sequentially. That is, a batch of rows are inserted or a batch of sequential rows are fetched. As a result, a larger page size typically results in better performance characteristics as fewer logical and/or physical page I/O requests are required to read a given amount of data. This is not always the case when the average temporary table row size is smaller than the page size divided by 255. A maximum of 255 rows can exist on any page regardless of the page size. For example, a query that requires a temporary table with fifteen-byte rows would be better served with a 4 KB temporary table space page size because 255 such rows can all be contained within a 4 KB page. An 8 KB (or larger) page size would result in at least 4 KB (or more) bytes of wasted space on each temporary table page; and therefore would not reduce the number of I/O requests required.
- If more than fifty percent of the regular table spaces in your database use the same page size, it can be advantageous to define your temporary table spaces with the same page size. The reason for the advantage is that this arrangement enables your temporary table space to share the same buffer pool space with most or all of your regular table spaces. This, in turn, simplifies buffer pool tuning.
- When reorganizing a table using a temporary table space, the page size of the temporary table space must match that of the table. For this reason, you should ensure there are temporary table spaces defined for each different page size used by existing tables that you may reorganize using a temporary table space.

  Note: You can also perform reorganization without a temporary table space by reorganizing the table “inplace”; that is, directly in the target table space. Of course, this “inplace” reorganization requires that there be extra space in the target table space for the reorganization process. Refer to *Administration Guide, Performance* for additional information on reorganization of tables.
- In general, when temporary table spaces of differing page sizes exist, the optimizer will most often choose the temporary table space with the largest buffer pool. In such cases, it is often wise to assign an ample buffer pool to one of the temporary table spaces, and leave any others with a smaller buffer pool. Such a buffer pool assignment will help ensure efficient utilization of main memory. For example, if your catalog table space uses 4 KB pages, and the remaining table spaces use 8 KB pages, the best temporary table space configuration may be a single 8 KB temporary table space with an ample buffer pool; and a single 4 KB table space with a small buffer pool.

Note: Catalog table spaces are restricted to use the 4 KB page size. As such, the database manager always enforces the existence of a 4 KB temporary table space to enable catalog table reorganizations.

- There is generally no advantage to defining more than one temporary table space of any single page size.
- SMS is almost always a better choice than DMS for temporary table spaces because:
  - Disk space is allocated on demand in SMS, whereas it must be pre-allocated in DMS. Preallocation can be a difficulty as shown in the following example: Temporary table spaces hold transient data that can have a very large peak storage requirement but a much lower average storage requirement. With DMS, the peak storage requirement must be pre-allocated, whereas with SMS, the extra disk space can be used for other purposes during off-peak hours.
  - The database manager does its best to keep temporary table pages in memory, and to avoid having them out on disk. As a result, the performance advantages of DMS are less significant.
  - SMS containers can take advantage of file system buffering; DMS containers cannot.

Recommendations for Catalog Table Spaces

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, and the page size for the table space. 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.

The considerations for determining the page size for a table space are as follows:

- For OLTP applications that perform random row reads and writes, a smaller page size is usually preferable, because it wastes less buffer pool space with unwanted rows.
- For DSS applications that access large numbers of consecutive rows at a time, a larger page size is usually better because it reduces the number of I/O requests that are required to read a specific number of rows. There is, however, an exception to this. If your row size is smaller than `pagesize/255`, there will be wasted space on each page (there is a maximum of 255 rows per page). In this situation, a smaller page size may be more appropriate.
- Larger page sizes may allow you to reduce the number of levels in the index.
- Larger pages support rows of greater length.
- On the default 4 KB page size, tables are restricted to 500 columns while the larger page sizes (8 KB, 16 KB, and 32 KB) support 1012 columns.
- The maximum possible size of the table space is proportional to the page size of the table space. The limits are documented in the *SQL Reference*.

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 64 GB of regular table data, 64 GB of index data and 2 TB of long data.

Note: If you are using 8 KB pages, the table data and index data can be as much as 128 GB. If you are using 16 KB pages, the table data and index data can be as much as 256 GB. If you are using 32 KB pages, the table data and index data can be as much as 512 GB.

- 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. Refer to "Performance Considerations for DMS Devices" in the *Administration Guide, Performance* for more information.

Optimizing Performance When Data is Placed on RAID Devices

This section describes how to optimize performance when data is placed on Redundant Array of Independent Disks (RAID) devices. In general, you should do the following for each table space that uses a RAID device:

- Define a single container for the table space (using the RAID device).
- Make the EXTENTSIZE of the table space equal to, or a multiple of, the RAID stripe size.
- Ensure that the PREFETCHSIZE of the table space is:
  - the RAID stripe size multiplied by the number of RAID parallel devices (or a whole multiple of this product), and
  - a multiple of the EXTENTSIZE.
- Use the DB2_PARALLEL_IO registry variable (described below) to enable parallel I/O for the table space
- Use the DB2_STRIPED_CONTAINERS registry variable (described below) to ensure extent boundaries are aligned in the table space.

DB2_PARALLEL_IO

When reading data from, or writing data to table space containers, DB2 may use parallel I/O if the number of containers in the database is greater than 1. However, there are situations when it would be beneficial to have parallel I/O

enabled for single container table spaces. For example, if the container is created on a single RAID device that is composed of more than one physical disk, you may want to issue parallel read and write calls.

To force parallel I/O for a table space that has a single container, you can use the DB2_PARALLEL_IO registry variable. This variable can be set to "*" (asterisk), meaning every table space, or it can be set to a list of table space IDs separated by commas. For example:

```
db2set DB2_PARALLEL_IO=*        {turn parallel I/O on for all table spaces}
db2set DB2_PARALLEL_IO=1,2,4,8  {turn parallel I/O on for table spaces 1, 2,
                                 4, and 8}
```

After setting the registry variable, DB2 must be stopped (db2stop), and then restarted (db2start), for the changes to take effect.

DB2_STRIPED_CONTAINERS

Currently when creating a DMS table space container (device or file), a one-page tag is stored at the beginning of the container. The remaining pages are available for data storage by DB2, and are grouped into extent-sized blocks.

When using RAID devices for table space containers, it is suggested that the table space be created with an extent size that is equal to, or a multiple of, the RAID stripe size. However, because of the one-page container tag, the extents will not line up with the RAID stripes, and it may be necessary during an I/O request to access more physical disks than would be optimal.

DMS table space containers can now be created in such a way that the tag exists in its own (full) extent. This avoids the problem described above, but it requires an extra extent of overhead within the container. To create containers in this fashion, you must set the DB2 registry variable DB2_STRIPED_CONTAINERS to "ON", and then stop and restart your instance:

```
db2set DB2_STRIPED_CONTAINERS=ON
db2stop
db2start
```

Any DMS container that is created (with CREATE TABLESPACE or ALTER TABLESPACE) will have new containers with tags taking up a full extent. Existing containers will remain unchanged.

To stop creating containers with this attribute, reset the variable, and then stop and restart your instance:

```
db2set DB2_STRIPED_CONTAINERS=
db2stop
db2start
```

The Control Center and the LIST TABLESPACE CONTAINERS command will not show whether a container has been created as striped or not. They will continue to use ″file″ or ″device″, depending on how the container was created. To verify that a container was created as striped, you can use the /DTSF option of DB2DART to dump table space and container information, and look at the type field for the container in question. Also, the query container APIs, sqlbftcq( ) and sqlbtcq( ), can be used to create a simple application that will display the type.

Definitions for these new types have been added to the sqlutil.h header file:

```
#define SQLB_CONT_STRIPED_DISK 5      /* DMS: Striped disk */
#define SQLB_CONT_STRIPED_FILE 6      /* DMS: Striped file */
```

Federated Database Design Considerations

When designing a federated database, consider the following design topics:

- Space requirements
- Network prioritization

Typically, the data accessible from a federated database is not stored at that database. References to data source tables and views are stored within the system catalog, but the actual data is located at the data source. As such, you might need less storage space than a typical database. This general rule might not apply if your queries, due to collating system differences or lack of function at a data source, must be executed locally. In this case, tables are materialized at DB2 for processing.

Because the majority of federated system data is typically located at one or more data sources located across a network, consider changing the resources assigned to DB2 and your network system. You might see performance increases by allocating more resources to the network at the DB2 system than to the database manager itself.

Chapter 4. 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, wrappers, servers, nicknames, type mappings, function mappings, 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, refer to the *SQL Reference* manual. For information on command line processor commands and user APIs, refer to the *Command Reference* and *Administrative 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"
- "Starting DB2 UDB on Windows NT" on page 101
- "Using Multiple Instances of the Database Manager" on page 102
- "Organizing and Grouping Objects by Schema" on page 103
- "Enabling Intra-Partition Parallelism" on page 104
- "Enabling Data Partitioning" on page 104

Starting and Stopping DB2

You may need to start or stop DB2 during normal business operations; for example, you must start an instance before you can perform the following tasks:

- Connect to a database on the instance.
- Precompile an application.
- Bind a package to a database.
- Access host databases.

To start a DB2 instance on your system:

1. Log in with a user ID or name that has SYSADM, SYSCTRL, or SYSMAINT authority on the instance; or, log in as the instance owner.
2. On UNIX operating systems, run the start up script as follows:

   ```
   . INSTHOME/sqllib/db2profile      (for Bourne or Korn shell)
   source INSTHOME/sqllib/db2cshrc   (for C shell)
   ```

   where INSTHOME is the home directory of the instance you want to use.
3. Use one of these methods to start the instance:
   - From the Control Center, click with the right mouse button on the instance that you want to start and select the **Start** option from the pop-up menu.
   - From a command line, enter the **db2start** command.

   Note: The **db2start** command starts the instance according to the rules in "Setting the Current Instance" on page 113.

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

1. Log in or attach to an instance with a user ID or name that has SYSADM, SYSCTRL, or SYSMAINT authority on the instance; or, log in as the instance owner.
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, or SYSMAINT authority for this.
3. Force all applications and users off the database. You require SYSADM or SYSCTRL authority to force users.
4. On UNIX operating systems, run the start up script as follows:

   ```
   . INSTHOME/sqllib/db2profile      (for Bourne or Korn shell)
   source INSTHOME/sqllib/db2cshrc   (for C shell)
   ```

   where INSTHOME is the home directory of the instance you want to use.
5. Use one of these methods to start the instance:
   - From the Control Center, click with right mouse button on the instance that you want to stop and select the **Stop** option from the pop-up menu.
   - Stop the DB2 instance from the command line by typing the command:

     ```
     db2stop
     ```

The *db2stop* 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.

Note: If command line processor sessions are attached to an instance, you must run the **terminate** command to end each session before running the **db2stop**command. The **db2stop** command stops the instance defined by the DB2INSTANCE environment variable.

Starting DB2 UDB on Windows NT

The db2start command will launch DB2 as an NT Service. DB2 on Windows NT can still be run as a process by specifying the ″/D″ switch when invoking DB2START. DB2 can also be started as a Service using the Control Panel or ″NET START″ command.

In order to successfully launch DB2 as a service from DB2START, the user account must have the correct privilege as defined by the Windows NT operating system to start an NT Service. The user account can be a member of the Administrators, Server Operators, or Power Users group.

When running in a partitioned database environment, each database partition server is started as an NT service.

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:

```
db2 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:

```
db2 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. Refer to the *Quick Beginnings* guide appropriate to your platform for information on defining multiple DB2 instances on one machine.

Organizing and Grouping Objects by Schema

Database object names may be made up of a single identifier or they may be *schema qualified objects* made up of two identifiers. The schema, or high-order part, of a schema qualified object provides a means to classify or group objects in the database. 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.

Explicit use of the schema occurs when you use the high-order part of a two-part object name when referring to that object in a statement. For example, USER A issues a CREATE TABLE statement in schema C as follows:

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

Implicit use of the schema occurs when you do not use the high-order part of a two-part object name. When this happens, the CURRENT SCHEMA special register is used to identify the schema name used to complete the high-order part of the object name. The initial value of CURRENT SCHEMA is the authorization ID of the current session user. If you wish to change this during the current session, you can use the SET SCHEMA statement to set the special register to another schema name. Refer to the *SQL Reference* for more information.

As described in "Definition of System Catalog Tables" on page 148, some objects are created within certain schemas when the database is created.

In dynamic SQL statements, a schema qualified object name implicitly uses the CURRENT SCHEMA special register value as the qualifier for unqualified object name references. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified database object names.

Before creating your own objects, you need to consider whether you want to create them in your own schema or by using a different 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 157.

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 multi-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, refer to the *Administration Guide, Performance*.

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, refer to the *Command Reference* and *Administrative 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. All federated database objects (wrappers, servers, nicknames, etc.) are stored in the system catalog tables at this node.

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, refer to the *Administration Guide, Performance*.

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
- Creation of the 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 2. Designing Your Logical Database" on page 29. To find out more about physical database design, see "Chapter 3. Designing Your Physical Database" on page 55.

Create an Instance

An instance is a logical database manager environment where you catalog databases and set configuration parameters. Depending on your needs, you can create more than one instance. You can use multiple instances to do the following:

- Use one instance for a development environment and another instance for a production environment.
- Tune an instance for a particular environment.
- Restrict access to sensitive information.
- Control the assignment of SYSADM, SYSCTRL, and SYSMAINT authority for each instance.
- Optimize the database manager configuration for each instance.
- Limit the impact of an instance failure. In the event of an instance failure, only one instance is affected. Other instances can continue to function normally.

It should be noted that multiple instances have some minor disadvantages:

- Additional system resources (virtual memory and disk space) are required for each instance.
- More administration is required because of the additional instances to manage.

The instance directory stores all information that pertains to a database instance. You cannot change the location of the instance directory once it is created. The directory contains:

- The database manager configuration file
- The system database directory

- The node directory
- The DB2 diagnostic file (db2diag.log)
- The node configuration file (on Windows NT)
- Any other files that contain debugging information, such as the exception/register dump or the call stack for the DB2 processes.

On UNIX operating systems, the instance directory is located in the `INSTHOME/sqllib` directory, where `INSTHOME` is the home directory of the instance owner.

In a partitioned database system, the instance directory is shared between all database partition servers belonging to the instance. Therefore, the instance directory must be created on a network share drive that all machines in the instance can access.

As part of your installation procedure, you create a default instance of DB2 called “DB2”. On UNIX, the default can be called anything you want within the naming rules guidelines. The instance name is used to set up the directory structure.

To support the immediate use of this instance, the following are set during installation:

- The environment variable DB2INSTANCE is set to “DB2”.
- The DB2 registry variable DB2INSTDEF is set to “DB2”.

On UNIX, the default can be called anything you want within the naming rules guidelines.

These settings establish “DB2” as the default instance. You can change the instance that is used by default, but first you have to create an additional instance.

Before using DB2, the database environment for each user must be updated so that it can access an instance and run the DB2 programs. This applies to all users (including administrative users).

On UNIX operating systems, sample script files are provided to help you set the database environment. The files are: `db2profile` for Bourne or Korn shell, and `db2cshrc` for C shell. These scripts are located in the `sqllib` subdirectory under the home directory of the instance owner. The instance owner or any user belonging to the instance’s SYSADM group can customize the script for all users of an instance. Alternatively, the script can be copied and customized for each user.

The sample script contains statements to:

- Update a user's PATH by adding the following directories to the existing search path: the bin, adm, and misc subdirectories under the sqllib subdirectory of the instance owner's home directory.
- Set the DB2INSTANCE environment variable to the instance name.

Setting the DB2 Environment Automatically

Note: This discussion only applies to the UNIX operating system environments.

By default, the scripts affect the user environment for the duration of the current session only. You can change the .profile file to enable it to run the db2profile script automatically when the user logs on using the Bourne or Korn shell. For users of the C shell, you can change the .login file to enable it to run the db2shrc script file.

Add one of the following statements to the .profile or .login script files:

- For users who share one version of the script, add:

```
. INSTHOME/sqllib/db2profile    (for Bourne or Korn shell)
source INSTHOME/sqllib/db2cshrc (for C shell)
```

  where INSTHOME is the home directory of the instance that you wish to use.
- For users who have a customized version of the script in their home directory, add:

```
. USERHOME/db2profile         (for Bourne or Korn shell)
source USERHOME/db2cshrc      (in C shell)
```

  where USERHOME is the home directory of the user.

Setting the DB2 Environment Manually

Note: This discussion only applies to the UNIX operating system environments.

To choose which instance that you want to use, enter one of the following statements at a command prompt. The period (.) and the space are required.

- For users who share one version of the script, add:

```
. INSTHOME/sqllib/db2profile    (for Bourne or Korn shell)
source INSTHOME/sqllib/db2cshrc (for C shell)
```

  where INSTHOME is the home directory of the instance that you wish to use.
- For users who have a customized version of the script in their home directory, add:

```
. USERHOME/db2profile         (for Bourne or Korn shell)
source USERHOME/db2cshrc      (in C shell)
```

where USERHOME is the home directory of the user.

If you want to work with more than one instance at the same time, run the script for each instance that you want to use in separate windows. For example, assume that you have two instances called test and prod, and their home directories are /u/test and /u/prod.

In window 1:

- In Bourne or Korn shell, enter:

  ```
  . /u/test/sqllib/db2profile
  ```

- In C shell, enter:

  ```
  source /u/test/sqllib/db2cshrc
  ```

In window 2:

- In Bourne or Korn shell, enter:

  ```
  . /u/prod/sqllib/db2profile
  ```

- In C shell, enter:

  ```
  source /u/prod/sqllib/db2cshrc
  ```

Use window 1 to work with the test instance and window 2 to work with the prod instance.

Note: Enter the **which db2** command to ensure that your search path has been set up correctly. This command returns the absolute path of the DB2 CLP executable. Verify that it is located under the instance's sqllib directory.

Creating Multiple Instances

It is possible to have more than one instance on a system. You may only work within one instance of DB2 at a time.

The instance owner and the group that is the System Administration (SYSADM) group are associated with every instance. The instance owner and the SYSADM group are assigned during the process of creating the instance. One user ID or username can be used for only one instance. That user ID or username is also referred to as the instance owner.

Each instance owner must have a unique home directory. All of the files necessary to run the instance are created in the home directory of the instance owner's user ID or username. If it becomes necessary to remove the instance owner's user ID or username from the system, you could potentially lose files associated with the instance and lose access to data stored in this instance. For

this reason, it is recommended that you dedicate an instance owner user ID or username to be used exclusively to run DB2.

The primary group of the instance owner is also important. This primary group automatically becomes the system administration group for the instance and gains SYSADM authority over the instance. Other user IDs or usernames that are members of the primary group of the instance owner also gain this level of authority. For this reason, you may want to assign the instance owner's user ID or username to a primary group that is reserved for the administration of instances. (Also, ensure that you assign a primary group to the instance owner user ID or username; otherwise, the system-default primary group is used.)

If you already have a group that you want to make the system administration group for the instance, you can simply assign this group as the primary group when you create the instance owner user ID or username. To give other users administration authority on the instance, add them to the group that is assigned as the system administration group.

To separate SYSADM authority between instances, ensure that each instance owner user ID or username uses a different primary group. However, if you choose to have a common SYSADM authority over multiple instances, you can use the same primary group for multiple instances.

If you have Administrative authority on OS/2, or you belong to the Administrative group on Windows NT, or you have root authority on UNIX platforms, you can create additional DB2 instances using the **db2icrt** command. The machine that you run the command on becomes the instance-owning machine (node zero/0). Ensure that you create instances on a machine where an Administration Server resides.

Note: You can choose to update an existing singe-partition instance to the multi-partition format using the **db2iupdt** command.

Create Instance Command

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 6. Controlling Database Access" on page 281.

To create an instance, perform the following steps:

1. Log on using a user ID or name that has Administrative authority or belongs to an Administrators group.
2. From a command prompt, run the **db2icrt** command:

   ```
   db2icrt <instance_name>
   ```

   When working with DB2 Extended Enterprise Environment, you will also need to declare that you are creating an instance that is a partitioned database system. This is done using –s eee on the command line.

   When working with UNIX operating systems, the **db2icrt** command has the following optional parameters:

   - –h or –?

     This parameter is used to display a help menu for the command.
   - –d

     This parameter sets the debug mode for use during problem determination.
   - –a AuthType

     This parameter specifies the authentication type for the instance. Valid authentication types are SERVER, CLIENT, DCS, or DCE. If not specified, the default is SERVER, if a DB2 server is installed. Otherwise, it is set to CLIENT.

     Notes:

     a. The authentication type of the instance applies to all databases owned by the instance.

     b. On UNIX operating systems, the authentication type DCE is not a valid choice.
   - –u FencedID

     This parameter is the user under which the fenced user-defined functions (UDFs) and stored procedures will execute. This is not required if you install the DB2 client or the DB2 Software's Developer Kit. For other DB2 products, this is a required parameter.

     Note: FencedID may not be "root" or "bin".
   - –p PortName

     This parameter specifies the TCP/IP service name or port number to be used. This value will then be set in the instance's database configuration file.
   - –s InstType

     Allows different types of instances to be created. Valid instance types are: ee, eee and client.

   Examples:

- To create an instance for a DB2 server, you can use the following command:

  ```
  db2icrt -u db2fenc1 db2inst1
  ```

- If you installed the DB2 Connect Enterprise Edition only, you can use the instance name as the Fenced ID also:

  ```
  db2icrt -u db2inst1 db2inst1
  ```

- To create an instance for a DB2 client, you can use the following command:

  ```
  db2icrt db2inst1 –s client
  ```

Optionally on Windows NT, you may also want to specify a different instance profile path. If you do not specify the path, the instance directory is created under the SQLLIB directory, and given the shared name DB2 concatenated to the instance name. Read and write permissions are automatically granted to everyone in the domain. Permissions can be changed to restrict access to the directory.

If you do specify a different instance profile path, you must create a shared drive or directory.

When working with DB2 for Windows NT, you will also need to declare the logon and account name and password of the DB2 Service. This is done using /u: followed by the username and password (comma separated) on the command line.

Optionally on Windows NT, you may also want to specify the TCP/IP port range for the FCM. This is done using /r: followed by the base port and end port numbers (comma separated) on the command line.

For example, on DB2 for Windows NT Extended Enterprise Edition, you could have the following example:

```
db2icrt inst1 –s eee
   /p:\\machineA\db2mpp
   /u:yourname,yourpwd /r:9010,9015
```

3. Optionally, create an Administration Server.

Note: The **db2icrt** command grants the username used to create the instance the following Windows NT user rights:

- Act as a part of the operating system
- Create a token object
- Increase quota
- Logon as a service
- Replace a process level token.

The instance requires these user rights to access the shared drive, authenticate the user account, and run DB2 as a Windows NT service.

You can change the location of the instance directory from DB2PATH using the DB2INSTPROF environment variable. You require write-access for the instance directory. If you want the directories created in a path other than DB2PATH, you have to set DB2INSTPROF **BEFORE** entering the **db2icrt** command.

Listing Instances

To get a list of all the instances that are available on a system, enter:

```
db2ilist
```

To determine which instance applies to the current session,

```
set db2instance
```

Or, on UNIX operating systems,

```
db2 get instance
```

Setting the Current Instance

When you run commands to start or stop an instance's database manager, DB2 applies the command to the current instance. DB2 determines the current instance as follows:

- If the DB2INSTANCE environment variable is set for the current session, its value is the current instance. To set the DB2INSTANCE environment variable, enter:

  ```
  set db2instance=<new_instance_name>
  ```

- If the DB2INSTANCE environment variable is not set for the current session, DB2 uses the setting for the DB2INSTANCE environment variable from the system environment variables. On Windows NT, system environment variables are set in System Environment. On Windows 95, they are set in the `autoexec.bat` file. On OS/2, they are set in the `config.sys` file.
- If the DB2INSTANCE environment variable is not set at all, DB2 uses the registry variable, DB2INSTDEF.

  To set the DB2INSTDEF registry variable at the global level of the registry, enter:

  ```
  db2set db2instdef=<new_instance_name> -g
  ```

Auto-Starting Instances

On UNIX operating systems, to enable an instance to auto-start after each system restart, enter the following command:

```
db2iauto -on InstName
```

where InstName is the login name of the instance.

On UNIX operating systems, to prevent an instance from auto-starting after each system restart, enter the following command:

```
db2iauto -off InstName
```

where InstName is the login name of the instance.

Running Multiple Instances Concurrently

You can start multiple DB2 instances as long as they belong to the same level of code.

To run multiple instances concurrently, use one of the following methods:

- From the Control Center, click with the right mouse button on another instance that you want to start and select the **Start** option.
- From the command line, do one of the following:
  1. Set the DB2INSTANCE variable to the name of the other instance that you want to start by entering:

     ```
     set db2instance=<another_instName>
     ```
  2. Start the instance by entering the **db2start** command.

License Management

The management of licenses for your DB2 products is done primarily through the License Center within the Control Center of the online interface to the product. From the License Center you can check the license information, statistics, registered users, and current users for each of the installed products.

Establish Environment Variables and the Profile Registry

Environment and registry variables 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 variables stored in the DB2 profile registries. Users with system administration (SYSADM) authority for a given instance can update registry values for that instance. Use the **db2set** command to update registry variables without rebooting; this information is stored immediately in the profile registries. The DB2 registry applies the updated information to DB2 server instances and DB2 applications started after the changes are made.

When updating the registry, changes do not affect the currently running DB2 applications or users. Applications started following the update use the new values.

Note: The DB2 environment variables DB2INSTANCE, DB2NODE, 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. On UNIX platforms, the export command may be used instead of the **set** command, and a system reboot is not necessary.

Using the profile registry allows for centralized control of the environment variables. "DB2 Registry and Environment Variables" in the *Administration Guide, Performance* lists many of the environment variables and registry variables. 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 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. Values defined in this level override their settings in the global level.
- 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. In DB2 UDB EEE, one global-level profile exists at each machine.
- 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. Values defined at this level override comparable settings in the instance and global levels.
- The DB2 Instance Profile 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. (Or the export command on UNIX platforms.)

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 global profile (using the db2set -G command as shown below).

Using the db2set Command

The **db2set** command supports the local declaration of the registry variables (and 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, use:

```
db2set -lr
```

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

```
db2set
```

To list all defined registry variables in the profile registry, use:

```
db2set -all
```

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

```
db2set registry_variable_name
```

To show the value of a registry variable at all levels, use:

```
db2set registry_variable_name -all
```

To delete a variable's value at a specified level, you can use the same command syntax to set the variable but specify nothing for the variable value. For example, to delete the variable's setting at the node level, enter:

```
db2set registry_variable_name= -I instance_name
node_number
```

To delete a variable's value and to restrict its use, if it is defined at a higher profile level, enter:

```
db2set registry_variable_name= -null instance_name
```

This command will delete the setting for the parameter you specify and restrict high level profiles from changing this variable's value (in this case,

DB2 global-level profile). However, the variable you specify could still be set by a lower level profile (in this case, the DB2 node-level profile).

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

```
db2set registry_variable_name=new_value
```

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

```
db2set registry_variable_name=new_value
   -I instance_name
```

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

```
db2set registry_variable_name=new_value -G
```

To set registry variables at the user level, use:

```
db2set -ul
```

To set registry variables at the user level for a specific user, use:

```
db2set -ul user_name
```

Notes:

1. The parameters ″-I″, ″-G″, and ″-ul″ cannot be used at the same time in the same command.
2. Some parameters will always default to the global level profile. They cannot be set at the instance or node level profiles; for example, `db2system` and `db2instdef`.
3. On UNIX, you must have system administration (SYSADM) authority to change registry values for an instance. Only users with root authority can change parameters in global-level registries.

To set the search scope value at the global level in LDAP, use:

```
db2set -gl db2ldap_search_scope = value
```

where the *value* can be “local”, “domain”, or “global”.

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

```
db2set registry_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 registry_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 registry_variable_name node_number
```

Setting Environment Variables on OS/2

It is strongly recommended that all DB2-specific registry variables be defined in the DB2 profile registry. If DB2 variables are set outside of the registry, remote administration of those variables is not possible, and the workstation must be rebooted in order for the variable values to take effect.

On OS/2, you should have no environment variables defined in config.sys apart from DB2PATH and DB2INSTPROF. All variables 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 registry variable defines the default instance name to use if DB2INSTANCE is not set.

DB2INSTANCE and DB2PATH are set when DB2 is installed; DB2INSTPROF can be set after installation. The environment variable DB2PATH must be set; this environment variable is set during installation and you should not modify it. Setting DB2INSTANCE and DB2INSTPROF environment variables is optional.

To determine the setting of an environment variable, enter:

```
set variable
```

To change the setting of an environment variable, enter the following command:

```
set variable=value
```

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%\DEFAULT.ENV
  ```
- 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 Instance Profile Registry is located under:

```
%DB2INSTPROF%\PROFILES.REG
```

Setting Environment Variables on Windows NT and Windows 95

It is strongly recommended that all DB2-specific registry variables be defined in the DB2 profile registry. If DB2 variables are set outside of the registry, remote administration of those variables is not possible, and the workstation must be rebooted in order for the variable values to take effect.

Windows 32-bit operating systems have one system environment variable, DB2INSTANCE, that can only be set outside the profile registry; however, you are not required to set DB2INSTANCE. The DB2 profile registry variable DB2INSTDEF may be set in the global level profile to specify the instance name to use if DB2INSTANCE is not defined.

DB2 Extended Enterprise Edition servers on Windows NT have two system environment variables, DB2INSTANCE and DB2NODE, that can only be set outside the profile registry. You are not required to set DB2INSTANCE. The DB2 profile registry variable DB2INSTDEF may be set in the global level profile to specify the instance name to use if DB2INSTANCE is not defined.

The DB2NODE environment variable is used to route requests to a target logical node within a machine. This environment variable must be set in the session in which the application or command is issued and not in the DB2 profile registry. If this variable is not set, the target logical node defaults to the logical node which is defined with port zero (0) on the machine.

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 and Windows 98: 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.

Note: The environment variable DB2INSTANCE can also be set at the session (process) level. For example, if you want to start a second DB2 instance called TEST, issue the following commands in a command window:

```
set db2instance=TEST
db2start
```

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.

DB2 UDB provides the capability of accessing DB2 UDB registry variables at the instance level on a remote machine. Currently, DB2 UDB registry variables are stored in three different levels: machine or global level, instance level, and node level. The registry variables stored at the instance level (including the node level) can be redirected to another machine by using

DB2REMOTEPREG. When DB2REMOTEPREG is set, DB2 UDB will access the DB2 UDB registry variables from the machine pointed to by DB2REMOTEPREG. 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

It is strongly recommended that all DB2-specific registry variables be defined in the DB2 profile registry. If DB2 variables are set outside of the registry, remote administration of those variables is not possible.

On UNIX operating systems, you must set the system environment variable DB2INSTANCE.

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.

Note: Except for PATH and DB2INSTANCE, all other DB2-supported variables must be set in the DB2 profile registry. To set variables that are not supported by DB2, define them in your script files, **db2profile**and **db2cshrc**.

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)" on page 124.)

- 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 Instance 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
```

Setting Environment Variables on Windows 3.x

The DB2 environment on Windows 3.x is not controlled by profile registries. Instead, Windows 3.x clients define environment keywords in the `db2.ini` file (typically found in the `C:\windows` directory).

On Windows 3.x, the parameters that control the DB2 environment are called environment keywords. However, because many Windows 3.x keywords are also used on operating systems that use the DB2 profile registries, environment keywords may also be referred to as registry variables.

The `db2.ini` initialization file is an ASCII file that stores values for the Windows 3.x client environment keywords. Within this file, there is just one section header.

The parameters are set by specifying a keyword with its associated keyword value in the form:

```
KEYWORD=keywordValue
```

For example, here is what would appear in a sample `db2.ini` file following the section header:

```
DB2PATH=C:\SQLLIB\WIN
DB2INSTANCE=DB2
DB2INSTPROF=C:\SQLLIB
DB2TRACEON=N
```

Notes:

1. All the keywords and their associated values must be located below the section header.
2. The keywords are not case sensitive; however, their values can be if the values are character-based.
3. Comment lines use a semicolon in the first position of a new line.
4. Blank lines are permitted. If duplicate entries for a keyword exist, the first entry is used (and no warning is given).

The `db2.ini` file is located in the Windows product directory.

On Windows 3.x, the Client for DB2 Version 4, Version 5, and Version 6 must set this information **only** in the `db2.ini` file.

DB2 Administration Server (DAS)

DB2 Administration Server (DAS) is a special DB2 administration control point used only to assist with administration tasks on other DB2 servers. You must have a running DAS if you want to use the Client Configuration Assistant (CCA) or the Control Center (CC). DAS assists the CC and CCA when working on the following administration tasks:

- Enabling remote administration of DB2 Servers.
- Providing the facility for job management, including the ability to schedule the execution of both DB2 and operating system command scripts. These command scripts are user-defined. The Control Center is used to define the schedule of jobs, view the results of completed jobs, and perform other administrative tasks against jobs located either remotely or locally to the DAS.
- Providing a means for discovering information about the configuration of DB2 instances, databases, and other DB2 Administration Servers in conjunction with the DB2 Discovery utility. This information is used by the Client Configuration Assistant (CCA) and the Control Center (CC) to simplify and automate the configuration of client connections to DB2 databases.

You can only have one DAS on a machine. DAS is configured during installation to start when the operating system is booted.

DAS is used to perform remote tasks on the host system on behalf of a client request from the Control Center or the Client Configuration Assistant. Authorized access to DAS requires clients with SYSADM authority. All of the clients can be part of the SYSADM_GROUP configuration parameter.

Some of the requested tasks may require specific authority to run. The DAS runs under the identifier of a specific user. The privileges granted to that user must be restricted to only those tasks or operations desired by the administrator, but provide sufficient authority to carry out all desired commands. Generally, the tasks or operations required include:

- Query the operating system (OS) configuration information.
- Query the OS for user and group information.
- Act against other DB2 instances to start or stop them.
- Execute scheduled jobs.
- Collect information for Connectivity and Protocol Configuration.

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

Creating the DAS

Typically, the setup program creates a DAS on the instance-owning machine during DB2 installation. If, however, the setup program failed to create it, you can manually create a DAS.

As an overview of what occurs during the installation process as it relates to DAS, consider the following:

- On the OS/2 or Windows NT platforms:

  Log on to the machine you want to create the DAS on using an account that has local Administrator authority. If a specific user is to be identified, create a user with local Administrator authority. Enter `db2admin create`. If a specific user account is desired, you must use "/USER:" and "/PASSWORD:" when issuing `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. Do not use the user ID or account name created for the DAS as a User Account. Set the password for the account name to "Password Never Expires". 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. Refer to the *Command Reference* for more information on this command.

  On DB2 UDB for Windows NT Extended Enterprise Edition, if you are using the CCA or the Control Center to automate connection configuration to a DB2 server, the database partition server that is on the same machine as the DAS will be the co-ordinator node. This means that all physical connections from the client to the database will be directed to the database partition server on the instance-owning machine before being routed to other database partition servers.

  On DB2 UDB for Windows NT Extended Enterprise Edition, creating additional Administration Servers on other machines allows the CCA or Control Center to configure other systems as co-ordinator nodes using DB2 Discovery. To do this, perform the following:

  1. Log on to the machine using an account that has local Administrator authority.
  2. Create a Windows NT account that has local Administrator authority that will be used by the DAS. Ensure that the username of the account adheres to the DB2 naming conventions. When creating the account for the DAS, note the following:
     - Do not use the account for the DAS as a User Account.
     - Set the password for the account to `Password Never Expires`.
  3. Run the following command:

     ```
     db2admin create /user:username
        /password:passwrd
     ```

where username and passwrd are the username and password for the DAS.

- 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
     ```

     – On AIX:

     ```
     /usr/lpp/db2_05_00&/instance/
        dasicrt ASname
     ```

     – On HP-UX, SCO Unixware 7 or Solaris:

     ```
     /opt/IBMdb2/V5,0/instance/
        dasicrt ASname
     ```

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

  Note: If you are running NIS and NIS+, you need to set up the user and group names in such a way that:
  – The primary group of the DAS must be in the secondary group of all the instances.
  – The secondary group of the DAS must contain the primary group of all the instances.

  Secondary group lists are modified automatically only if NIS and NIS+ is not running on the system.

  Because a user ID can only own one instance, you must have a separate user ID to own each DB2 Administration Server (DAS) that you create.

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

Starting and Stopping the DAS

To manually start or stop the DAS, you must first log on to the machine using an account or user ID that has local Administrator authority.

When working on DB2 for OS/2 or DB2 for Windows NT, you must do the following:

- To start the DAS, enter `db2admin start`
- To stop the DAS, enter `db2admin stop`

Note: For both cases under Windows NT, the person using these commands must have SYSADM, SYSCTRL, or SYSMAINT authority.

When working on DB2 for any of the UNIX operating systems, you must do the following:

- To start the DAS, perform the following steps:
  1. Log in as the DAS owner.
  2. Run the start up script using one of the following:

     ```
     . INSTHOME/sqllib/db2profile    (for Bourne or Korn shell)
     source INSTHOME/sqllib/db2cshrc (for C shell)
     ```

     where INSTHOME is the home directory of the instance.
  3. Start the DAS using the **db2admin** command as follows:

     ```
     db2admin start
     ```

  Note: The DAS is automatically started after each system reboot.
- To stop the DAS, perform the following steps:
  1. Log in as the DAS owner.
  2. Run the start up script using one of the following:

     ```
     . INSTHOME/sqllib/db2profile    (for Bourne or Korn shell)
     source INSTHOME/sqllib/db2cshrc (for C shell)
     ```

     where INSTHOME is the home directory of the instance.
  3. Stop the DAS using the **db2admin** command as follows:

     ```
     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.

Listing the DAS

To obtain the name of the DAS on your machine, use:

```
dasilist
```

This command is found in the bin subdirectory under the subdirectory specific to the installed DB2 version and release.

Configuring the DAS

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

```
db2 get admin cfg
```

This will show you the current values that were given as defaults during the installation of the product or those that were given during previous updates to the configuration parameters.

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

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

Refer to 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:

```
db2 reset admin cfg
```

Changes to the database manager configuration file become effective only after they are loaded into memory (that is, when a `db2admin stop` is followed by a `db2admin start`; or, in the case of a Windows NT platform, stopping and starting the service.)

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

Security Considerations for the DAS

You must first logon to the machine using an account or user ID that has local Administrator authority.

Note: On Windows NT, you should not use the **Services** utility in the **Control Panel** to change the logon account for the DAS since some of the required access rights will not be set for the logon account. Always use the **db2admin** command to set or change the logon account for the DB2 Administration Server (DAS).

After creating the DAS, you can set or change the logon account using the **db2admin** command as follows:

```
db2admin setid username password
```

where *username* and *password* are the username and password of an account that has local Administrator authority.

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

Updating the DAS

On UNIX operating systems, if DB2 is updated by installing a Program Temporary Fix (PTF) or a code patch, all DB2 Administration Servers (DAS) as well as all exiting instances should be updated. To update a DAS, use the

dasiupdt command available in the instance subdirectory under the subdirectory specific to the installed DB2 version and release.

You must first logon to the machine using an account or user ID that has local Administrator authority.

The command is used as follows:

```
dasiupdt InstName
```

The InstName is the login name of the instance owner. There are also optional parameters for this command that can be placed before the InstName and separated by spaces:

- –h or –?

  Displays a help menu for this command.
- –d

  Sets the debug mode, which is used for problem analysis.

Removing the DAS

You must first logon to the machine using an account or user ID that has local Administrator authority.

To remove the DAS:

- On the OS/2 or Windows NT operating systems:
  1. Stop the DAS, using db2admin stop.
  2. Backup (if needed) all the files in the db2das00 subdirectory under the sqllib subdirectory. The instance directory is indicated by the *DB2INSTPROF* registry variable.

     Note: This example assumes db2das00 is the name of the DAS to be removed.
  3. Drop the DAS, using db2admin drop.

  Note: Under Windows NT, the person using this command must have SYSADM, SYSCTRL, or SYSMAINT authority.
- On UNIX operating systems:
  1. Log in as the DAS owner.
  2. Run the start up script using one of the following:

     ```
     . INSTHOME/sqllib/db2profile    (for Bourne or Korn shell)
     source INSTHOME/sqllib/db2cshrc (for C shell)
     ```

     where INSTHOME is the home directory of the instance.
  3. Stop the DAS using the **db2admin** command as follows:

```
db2admin stop
```

4. Backup (if needed) all the files in the sqllib subdirectory under the home directory of the DAS. The instance directory is indicated by the *DB2INSTPROF* registry variable.
5. Log off.
6. Log in as root and remove the DAS using the **dasidrop** command as follows:

   ```
   dasidrop ASName
   ```

   where the ASName is the instance name of the Administration Server. This command is found in the instance subdirectory under the subdirectory specific to the installed DB2 version and release.

   Note: The **dasidrop** command removes the sqllib directory under the home directory of the DB2 Administration Server (DAS).

Setting Up DAS with EEE Systems

The following information shows the steps necessary to configure DB2 EEE servers (Sun, NT, and AIX) for remote administration using the Control Center (CC).

During installation, the setup program creates a single DAS on the instance-owning machine. You may want to create additional DAS on other machines to allow the Control Center (CC) or the Client Configuration Assistant (CCA) access to other coordinator nodes. The overhead of working as a coordinator node can then be spread to more than one node in an instance.

To distribute the coordinator function:

1. Create a new DAS on the selected additional machines in the partitioned database system.
2. Catalog each DAS as a separate system in the CC or CCA.
3. Catalog the same instance under each new system, and each time specify the same machine name used to catalog the DAS.

There are two (2) aspects to configuration: That which is required for the DB2 Administration Server (DAS), and that which is recommended for the target, administered DB2 instance. In the three sections which follow, a section is devoted to each of the two configuration topics. Each of the configuration topics is preceded by a section describing the assumed environment.

Example Environment:

Product/version:
DB2 UDB EEE V5.2

Install path:
install_path

TCP services file:
tcp_services_file

DB2 Instance:

name: db2inst

owner ID:
db2inst

instance path:
instance_path

Nodes:
3 nodes, db2nodes.cfg:

- 0 hostA 0 hostA0switch
- 1 hostA 1 hostA1switch
- 2 hostB 0 hostBswitch

DB name:
db2instDB

DAS:

name: db2as

owner/user ID:
db2as

instance path:
das_path

install/run host:
hostA

internode communications port:
16000 (unused port for hostA and hostB)

Note: Please substitute site-specific values for the above fields. For example, the following table contains example pathnames for each supported EEE platform:

Table 20. Example Pathnames for Each Supported EEE Platform

Paths	DB2 UDB EEE V5.2 for AIX	DB2 UDB EEE V5.2 for Solaris	DB2 UDB EEE V5.2 for Windows NT
install_path	/usr/lpp/db2_05_00	/opt/IBMdb2/V5.0	C:\sqllib
instance_path	/home/db2inst/sqllib	/home/db2inst/sqllib	C:\profiles\db2inst
das_path	/home/db2as/sqllib	/home/db2as/sqllib	C:\profiles\db2as
tcp_services_file	/etc/services	/etc/services	C:\winnt\system32 \drivers\etc\services

When installing DB2 UDB EEE, the setup program creates a DAS on the instance-owning machine. The database partition server resides on the same machine as the DAS and is the connection point for the instance. That is, this database partition server is the coordinator node for requests issued to the instance from the Control Center (CC) or the Client Configuration Assistant (CCA).

DAS Configuration: The DAS is an administrative control point which performs certain tasks on behalf of the Control Center (CC). There can be at most one (1) DAS per physical machine. In the case of an EEE instance which consists of several machines, at least one of the machines must be running a DAS so that the CC can administer the EEE instance. This DAS (db2as) "represents" the system that is present in the CC navigator tree as the parent of the target DB2 instance (db2inst).

For example, db2inst consists of three nodes distributed across two physical machines or hosts. The minimum requirement can be fulfilled by running db2das on either hostA **or** hostB.

Notes:

1. The number of partitions present on hostA does not have any bearing on the number of DASes that can be run on that host. You can run only one copy of db2as on hostA regardless of the multiple logical nodes (MLN) configuration for that host.
2. It is not necessary to create the DAS ID, db2as, on all hosts. Rather, it is necessary for it to exist only on the host upon which it is running. As well, it is not necessary for the home directory of the DAS ID to be mounted on all hosts. In particular with this example, the ID db2as must exist on hostA, is not required on hostB, and db2as's home directory does not need to be mounted on hostB.

Control Center Communications with DAS: Service Ports: The Control Center (CC) communicates with the DAS using a TCP service port, 523. Since this port is reserved for exclusive use by DB2 UDB, it is not necessary to insert new entries into the *tcp_services_file*.

Internode Administrative Communications: Service Ports: For some administrative tasks, the DAS must establish communications with all nodes. In order to do so, a named TCP port must be defined in the *tcp_services_file* for each host which participates in the instance.

Note: Windows NT EEE will attempt to add the TCP port entry into the *tcp_services_file* for you.

For example, db2inst is defined across two hosts, hostA and hostB. As specified in "Example Environment" on page 130, port 16000 is unused on both hosts. Therefore, the following line must be inserted into the *tcp_services_file* for both hostA and hostB.

```
db2ccmsrv  16000/tcp
```

The db2ccmsrv port name must be present, spelled exactly as shown above, and the same port number selected must be used on all hosts.

Internode Administrative Communications: UNIX DB2 EEE Servers: Once the TCP port line is inserted into the *tcp_services_file* on hostA and hostB, it is necessary to start an administrative listener process or daemon, db2cclist, on all hosts that participate in the instance. You can do so manually from the command line, or configure the system to automatically invoke db2cclst every time the system boots:

Manual:

From the ID of the instance you wish to administer, db2inst, invoke the following command from either hostA or hostB:

```
rah 'install_path/bin/db2cclst'
```

For example, on AIX this command invocation would appear as:

```
rah '/usr/lpp/db2_05_00/bin/db2cclst'
```

Automatic:

From an ID with Superuser privileges (like root) execute the following command on hostA and hostB:

```
mkitab "db2cclst::once:su - db2inst -c install_path /bin/db2cclst"
```

For example, on AIX this command invocation would appear as:

```
mkitab "db2cclst::once:su - db2inst -c install_path
   /usr/lpp/db2_05_00/bin/db2cclst"
```

Every time either machine boots, db2cclist is invoked without user intervention.

To verify that the listener daemon is active on each host, the following command can be invoked from the instance ID, db2inst:

```
rah 'ps -ef | grep db2cclst'
```

If you do not find the db2cclst process running on each host, additional diagnostic information can be obtained by adding the following line to /etc/syslog.conf on each host:

```
*.info  /tmp/db2/user.info
```

where the file /tmp/db2/user.info can be replaced with a more appropriate file.

Note: The file must exist and the SYSLOG daemon must be asked to re-read its configuration file after the changes are made:

```
kill -1 <syslogd PID>
```

where syslogd PID can be obtained by executing

```
ps -ef | grep syslogd
```

Then, after manually invoking the listener as described above, you can view the syslog file /tmp/db2/user.infoon the failing host for error messages generated by db2cclst.

Internode Administrative Communications: Windows NT DB2 EEE Servers: The DB2 Remote Command Service (db2rcmd.exe) automatically handles internode administrative communications. In the event that a failure does occur, the Windows NT registry will contain diagnostic information.

Security: In order for the DAS to perform some administrative tasks against an instance, it must possess sufficient authority. In particular, the DAS must be a System Administrator (SYSADM) for the target, administered instance.

It is necessary to grant the DAS such authority for all DB2 instances that it will administer. Candidate instances are those which are installed on the same machine as the DAS. For a DB2 EEE instance, at least one database partition server must be present on the same machine as the DAS for it to be eligible as described above.

For example on UNIX, one way in which db2as can be granted the required authority to administer db2inst is to ensure that the primary groups of db2inst and db2as are identical. Alternatively, it is sufficient to make the primary group of db2inst a secondary group of db2as, and the primary group of db2as a secondary group of db2inst. Finally, another option would be to set the SYSADM_GROUP database administration configuration parameter for db2inst to the primary group of db2as.

On Windows NT, db2as must be a member of the Local Administrators group on hostA and hostB. In addition to the option of creating the db2as ID and

adding it to the Local Administrators group on both hosts, one could create a domain ID for db2as and add this domain ID to the Local Administrators group on each host.

Environment: Installation for the DAS should configure certain registry variables that are necessary for proper operation. To verify the current values for these variables, execute the following command from either the DB2 instance ID, db2inst, or the DAS ID, db2das:

```
db2set -g
```

At least the following parameters must be defined with the following values:

```
DB2SYSTEM=hostA
DB2ADMINSERVER=db2as
```

As well, in order to communicate with the DAS from the Control Center (CC), ensure that the DB2COMM variable is set to TCPIP. To verify this setting, execute the following command from the DAS ID, db2as, and check at the global (`-g`) and instance (`-i`) levels (only one need be set):

```
db2set -all
```

Along the same lines, verify that the DB2COMM parameter is set to TCPIP for the DB2 instance to enable communications between the CC and db2inst by issuing the following command from the db2inst ID:

```
db2set -all
```

If you modify this parameter for the DAS, then you must restart the DAS for the change to take effect. Restart of the DB2 instance is also required if this parameter is modified for the DB2 instance. For db2inst, you would issue a *db2stop* followed by a *db2start*, whereas *db2admin stop* and *db2admin start* would be issued for the DAS.

Discovery of Administration Servers, Instances, and Databases: Known Discovery allows you to discover instances and databases on systems that are known to your client, and add new systems so that their instances and databases can be discovered. Search Discovery provides all of the facilities of Known Discovery and adds the option to allow your local network to be searched for other DB2 servers.

To have a server support Known Discovery, set the *discover* parameter in the DAS configuration file to `KNOWN`. To have it support Search Discovery, set this parameter to `SEARCH`. To prevent discovery of a server, and all of its instances and databases, set this parameter to `DISABLE`.

Note: The TCP/IP host name returned to a client by Search Discovery is the same host name that is returned by your DB2 server system when you enter the **hostname** command. On the client, the IP address that this

host name maps to is determined by either the TCP/IP domain name server (DNS) configured on your client machine or, if a DNS is not configured, a mapping entry in the client's `hosts` file. If you have multiple adapter cards configured on your DB2 server system, you must ensure that TCP/IP is configured on the server to return the correct hostname, and that the DNS or local client's `hosts` file, maps the hostname to the IP address desired.

On the client, enabling Discovery is also done using the *discover* parameter; however, in this case, the *discover* parameter is set in the client instance (or server acting as a client) as follows:

- **Known**

  Allows the CCA to refresh systems in the known list, and to add new systems to the list by using the **Add Systems** push button. When the *discover* parameter is set to `KNOWN`, the CCA will not be able to search the network.
- **Search**

  Enables all of the facilities of Known Discovery, and enables network searching.
- **Disable**

  Disables Discovery. In this case, the **Search the network** option is not available in the "Add Database SmartGuide".

Note: The *discover* parameter defaults to `SEARCH` on all client and server instances. The *discover* parameter defaults to `SEARCH` on all DB2 Administration Servers (DAS) except DAS installed in a UNIX Extended Enterprise Edition environment, where *discover* defaults to `KNOWN`.

Additional Settings for Search Discovery: Search Discovery requires that the *discover_comm* parameter be set on both the server (in the DB2 Administration Server's configuration file) and the client (in the database manager configuration file).

The *discover_comm* parameter is used to control the communications protocols that the server will listen to for search requests from clients, and that clients will use to send out search requests. The *discover_comm* parameter can be set to TCP/IP or NetBIOS. Only these protocols are currently supported.

On the DAS, the values specified for *discover_comm* must be equal to, or a subset of, the values set for *db2comm*.

Note: To avoid problems with the Control Center and the Client Configuration Assistant, ensure that the *db2comm* parameter is set in the

DB2 registry using the **db2set** command. It is not recommended that you use any other method to set the *db2comm* parameter.

On the server, the *discover_comm* parameter is set in the DAS configuration file. On the client (or a server acting as a client), *discover_comm* is set in the database manager configuration file.

Note: When using Search Discovery, at least one protocol specified by the *discover_comm* parameter on the client must match those specified by the *discover_comm* parameter on the DAS. If there is no match, the server will not respond to the client's requests.

To check the settings for the *db2comm* registry variable use the following:

```
db2set db2comm
```

In addition, there are two DB2 profile registry variables that can be used to tune Search Discovery via NetBIOS on the client: *db2discoverytime* and *db2nbdiscoveryrcvbufs*. the default values should be suitable in most cases.

Hiding Server Instances and Databases from Discovery: You may have multiple instances, and multiple databases within these instances, on a server. You may want to hide some of these from the Discovery process.

To allow clients to discovery server instances on a system, set the *discover_inst* database manager configuration parameter in each server instance on the system to `ENABLE` (this is the default value). Set this parameter to `DISABLE` to hide this instance and its databases from Discovery.

To allow a database to be discovered from a client, set the *discover_db* database configuration parameter to `ENABLE` (this is the default value). Set this parameter to `DISABLE` to hide the database from Discovery.

Setting Discovery Parameters: The *discover* and *discover_comm* parameters are set in the DAS configuration file on the server system, and in the database manager configuration file on the client. Set the parameters as follows:

- On the DAS:

  Update the DAS configuration file using the command process:

  ```
  update admin cfg using discover [ DISABLE | KNOWN |
     SEARCH ]
  update admin cfg using discover_comm [ NETBIOS | TCPIP ]
  ```

  Stop and restart the DAS by entering the following commands:

  ```
  db2admin stop
  db2admin start
  ```

  Note: Search Discovery will only operate on NetBIOS and TCPIP.

- On the client:
  1. Start the Client Configuration Assistant (CCA).
  2. Click on the **Client Settings** push button.
  3. Select the **Communications** tab.
  4. Select the parameters that you want to modify from the **Parameters** window.
  5. Select a value for the parameter that you want to modify from the **Value** box.
  6. Click on the **OK** push button to close the **Client Settings** windows. A DB2 message window opens.
  7. Click on the **OK** push button and restart your applications so that your changes can take effect.

  Note: If the *discover_comm* includes NETBIOS, you must ensure that the Workstation name (*nname*) parameter is set for both the client and the DAS. Also, you must ensure that the *db2nbadapters* registry variable is set to the Adapter number that you want to use.

  Use the Control Center to set the *discover_inst* and *discover_db* parameters. Set the parameters as follows:
  1. Start the Control Center.
  2. Click on the **[+]** sign beside the **Systems** icon to get a list of systems.
  3. Click on the **[+]** sign beside a system icon to get a list of instances on that system.
  4. Click on the Control Center.
  5. Select the instance that you want to configure and click on the right mouse button.
  6. Select the **Configure** option from the pop-up menu. The Configuration window opens.
  7. Select the **Environment** tab and select the *discover_inst* parameter from the **Parameters** box.
  8. Enter the desired value in the **Values** box and click on **OK**.
  9. Click on the **[+]** sign beside an instance icon to get a list of databases in that instance.
  10. Select the database that you want to configure and click on the right mouse button.
  11. Select the **Configure** option from the pop-up menu. The Configuration Database window opens.
  12. Select the **Environment** tab and select the *discover_db* parameter from the **Parameters** box.
  13. Enter the desired value in the **Values** box and click on **OK**.

The *db2discoverytime* and *db2nbdiscoverrcvbufs* profile registry variables are set in the client instance (or a server acting as a client). Set the registry variables as follows:

– To set the *db2discoverytime* registry value to 60 seconds, enter the following command:

```
db2set db2discoverytime=60
```

This specifies that Search Discovery should wait 60 seconds for a response from servers.

– To set the *db2nbdiscoverrcvbufs* registry value to 20, enter the following command:

```
db2set db2nbdiscoverrcvbufs=20
```

This specifies the number of NetBIOS buffers that will be allocated for concurrent response messages from discovered servers.

Setting Up the DAS to Use the CCA and the Control Center

You must configure DB2 Discovery to retrieve information about systems on your network. DB2 Discovery is a feature that is used by the Client Configuration Assistant (CCA) and Control Center. Configuring for this feature may require you to update instance lists and the DB2 Administration Server (DAS) configuration to ensure that DB2 Discovery retrieves the correct information.

Update Instance Lists: A DB2 Administration Server (DAS) may not be aware of all the instances in a partitioned database system because initially when an instance is created, only the DAS on the instance-owning machine is aware of the instance.

If you created an instance on a machine that does not have a DAS, you can create a DAS on this machine to make the instance known.

Perform the following steps if you created more than one DAS, and you want each DAS to be aware of all the instances in your partitioned database system:

1. **For each DAS**

   Run the **db2ilist** command on the Administration Server machine to display a list of instances known to this DAS.

   Note: If the list of instances is complete, you do not need to carry out the remaining steps but can proceed to the next section.

2. **For each instance that is missing from the instance list in the previous step**

On the instance-owning machine, run the **db2nlist** command to see if there is an entry for the machine that has the DAS. If there is not, you must run the **db2ncrt** command to add this machine to the instance.

Note: The network shared drive for the instance must be available on the DAS machine.

Update the DAS Configuration

By default, the setup program sets the DB2SYSTEM registry variable to the Windows NT computer name. The system names that are retrieved by Discovery are the systems on which a DB2 Administration Server (DAS) resides. Discovery uses these systems as co-ordinator nodes when connections are established.

There are two ways of updating a DAS configuration:

- If you want to be able to select a co-ordinator node from a list of DB2 systems, set DISCOVER=SEARCH (which is the default) in each DB2 Administration Server's configuration file.

  When there are multiple DAS present, the same instance may appear in more than one system on the CCA or Control Center's interface; however, each system will have a different communications access path to instances. Users can select different DB2 systems as co-ordinator nodes for communications and thereby redistribute the workload.
- If you do not want users to be able to select the co-ordinator node, set DISCOVER=KNOWN on all DAS, except set DISCOVER=SEARCH on just one DAS in the DAS configuration. Discovery uses the database partition server where the DAS resides as a co-ordinator node when connections are established.

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.

Windows NT Considerations: If you are using DB2 Enterprise - Extended Edition on Windows NT, the node configuration file is created for you when you create the 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, refer to "Stored Procedures" in *Administration Guide, Performance*.) 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.

Creation of the Database Configuration File

A *database configuration file* is also created for each database. The creation of this file is done for you. 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 "Configuring DB2" found in *Administration Guide, Performance*.

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

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.

Replicating Configuration Information Using Response Files

A response-file generator utility called *db2rspgn* is available to create a response file that can be used when re-installing your system or when you wish to replicate to identical system the registry variables, database manager configuration parameters, and database administration configuration parameters of your current system.

After having installed a system with one or more DB2 products, and after tuning parameters for the environment, you can use *db2rspgn* to generate the required values into a response file. The response file can then be used to re-create the identical system.

The command line syntax declares the destination directory for the response file(s) and any supporting files. In addition, you can optionally specify the instances you wish copied; and, you can optionally disable the administration instance and/or the DataLinks server instance.

Refer to the appropriate *Quick Beginnings* to see the details on the syntax of this utility and a discussion on how to use the generated response files.

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.

Windows NT Considerations

If you are using DB2 Enterprise - Extended Edition in the Windows NT environment, the TCP/IP port range is automatically added to the services file by:

- The install program when it creates the instance or adds a new node
- The DB2ICRT utility when it creates a new instance
- The DB2NCRT utility when it adds the first node on the machine.

For additional information, refer to the *DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings*.

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:

- “Cataloging a Database” on page 152
- “Creating Nodegroups” on page 151
- “Creating a Table Space” on page 153
- “Creating a Schema” on page 157
- “Creating and Populating a Table” on page 158
- “Creating a Trigger” on page 174
- “Creating a User-Defined Function (UDF)” on page 176
- “Creating a User-Defined Type (UDT)” on page 179
- “Creating a View” on page 182
- “Creating a Summary Table” on page 187
- “Creating an Alias” on page 189
- “Creating a Wrapper” on page 190
- “Creating a Server” on page 191
- “Creating a Nickname” on page 198
- “Creating an Index or an Index Specification” on page 200.

For additional information related to the physical implementation of your database, see “Chapter 3. Designing Your Physical Database” on page 55.

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 102. 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: Refer to 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 148)
- 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 55). 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 153.

Before creating your database, see "Designing and Choosing Table Spaces" on page 75.

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, refer to the *SQL Reference* manual.
- A set of read-only views for the system catalog tables is created in the SYSCAT schema. Refer to "Catalog Views" in the *Administration Guide, Performance* 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. Refer to "Updatable Catalog Views" in the *Administration Guide, Performance* for information about these views.

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 331.

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 152.

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 104.

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.

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 E. Using Distributed Computing Environment (DCE) Directory Services" on page 699 for more information.

Lightweight Directory Access Protocol (LDAP) Directory Services

Lightweight Directory Access Protocol (LDAP) is an industry standard access method to directory services. Each instance of the database server will publish its existence and provide the protocol communication information in the LDAP directory. When a client connects to the database server, the communication information for the server can be retrieved from the LDAP

directory. Each client is no longer required to store the server connection information by cataloging a node entry locally on each machine. Instead, when a database is created, the database publishes its existence using the LDAP directory. Client applications search the LDAP directory for the database location and the information required to connect to the database.

LDAP is not a prerequisite for running DB2, but if you are operating in an LDAP environment, see "Appendix N. Lightweight Directory Access Protocol (LDAP) Directory Services" on page 829 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 69.
- 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, refer to 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 75 for information.)

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 9. Recovering a Database" on page 365.

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. Refer to 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, refer to the *Quick Beginnings* manual appropriate to your platform.

For information on the Distributed Computing Environment (DCE) cell directory, see "DCE Directory Services" on page 150 and "Appendix E. Using Distributed Computing Environment (DCE) Directory Services" on page 699.

Note: To improve performance, you may cache directory files, including the database directory, in memory. (Refer to "Directory Cache Support" in the *Administration Guide, Performance* 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, refer to "Directory Cache Support" in the *Administration Guide, Performance*.

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.

See "Designing and Choosing Table Spaces" on page 75 for design information on table spaces.

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 75.

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 55).

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 4 KB pages. In addition, declare a device for each database partition of 40 000 4 KB 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, refer to “Tuning Application Performance” in the *Administration Guide, Performance.*

DB2 can greatly improve the performance of sequential I/O using the sequential prefetch facility, which uses parallel I/O. Refer to “Understanding Sequential Prefetching” in the *Administration Guide, Performance* for details on this facility.

You also have the ability to create a table space that uses a page size larger than the default 4 KB size. The following SQL statement creates an SMS table space on a UNIX-based system with an 8 KB page size.

```
CREATE TABLESPACE SMS8K
   PAGESIZE 8192
   MANAGED BY SYSTEM
   USING ('FSMS_8K_1')
   BUFFERPOOL BUFFPOOL8K
```

Notice that the associated buffer pool must also have the same 8 KB page size.

The created table space cannot be used until the buffer pool it references is activated.

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, refer to "Understanding Sequential Prefetching" in the *Administration Guide, Performance.*

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.

RAW I/O

DB2 Universal Database supports direct disk access (raw I/O). This allows you to attach a direct disk access (raw) device to any DB2 Universal Database system. (The only exception is the Linux platform.) The following list demonstrates the physical and logical methods for identifying this type of device:

- To open a physical hard drive for direct disk access, use the following naming convention:

  ```
  \\.\PhysicalDriveN
  ```

  where N represents one of the physical drives in the system. For example, N could be replaced by 0, 1, 2, or any other positive integer.
- To open a logical raw partition (that is, an unformatted partition) use the following naming convention:

  ```
  \\.\N:
  ```

  where N: represents a logical drive letter in the system. For example, N: could be replaced by E: or any other drive letter on the system.

For example:

- On Windows NT, `\\.\d:` or `\\.\PhysicalDisk5`

  Note: You must have Windows NT Version 4.0 with Service Pack 3 installed to be able to write logs to a device.

- On UNIX-based platforms, /dev/rdblog8

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, the only schema they can create is one that has the same name as their own authorization ID.

Setting a Schema

You may wish to establish a default schema for use by unqualified object references in dynamic SQL statements issued from within a specific DB2 connection. This is done by setting the special register CURRENT SCHEMA to the schema you wish to use as the default. Any user can set this special register: No authorization is required.

The syntax of the SET SCHEMA statement is described in detail in the *SQL Reference* manual.

The following is an example of how to set the CURRENT SCHEMA special register:

```
SET CURRENT SCHEMA = 'SCHEMA01'
```

This statement can be used from within an application program or issued interactively. Once set, the value of the CURRENT SCHEMA special register is used as the qualifier (schema) for unqualified object references in dynamic SQL statements, with the exception of the CREATE SCHEMA statement where an unqualified reference to a database object exists.

The initial value of the CURRENT SCHEMA special register is equal to the authorization ID of the current session user.

Creating and Populating 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 691.

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*.

Details concerning the movement of data into and out of tables is presented in *Data Movement Utilities Guide and 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. Refer to the *SQL Reference* for more information.

Note: The maximum of 500 columns is true when using a 4 KB page size. The maximum is 1012 columns when using an 8 KB, 16 KB, or 32 KB page size.

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 694 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 179.

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
```

When creating a table, you can choose to have the columns of the table based on the attributes of a structured type. Such a table is called a “typed table”.

A typed table can be defined to inherit some of its columns from another typed table. Such a table is called a “subtable”, and the table from which it inherits is called its “supertable”. The combination of a typed table and all its subtables is called a “table hierarchy”. The topmost table in the table hierarchy (the one with no supertable) is called the “root table” of the hierarchy.

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

- “Large Object (LOB) Column Considerations”
- “Defining a Unique Constraint” on page 162
- “Defining Referential Constraints” on page 163
- “Defining a Table Check Constraint” on page 166
- “Creating a User-Defined Structured Type” on page 180
- “Creating a Typed Table” on page 167
- “Populating a Typed Table” on page 169
- “Creating a Table in Multiple Table Spaces” on page 172
- “Creating a Table in a Partitioned Database” on page 173.

You can also create a table that is defined based on the result of a query. This type of table is called a *summary table*. For more information, see “Creating a Summary Table” on page 187.

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 use LOB columns and may take up more space than in previous versions.

3. Do you want better performance for LOB columns, including those LOB columns in the DB2 system catalogs?

   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.

Defining Constraints

This section discusses how to define constraints:

- "Defining a Unique Constraint"
- "Defining Referential Constraints" on page 163
- "Defining a Table Check Constraint" on page 166.

For more information on constraints, see "Planning for Constraint Enforcement" on page 44 and refer to 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. However, a unique constraint may not be defined on a subtable.

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:

- The IMPORT utility always extends indexes incrementally, as opposed to LOAD, which always completely rebuilds them.
- As a result, when preforming an initial load of an empty table with indexes, LOAD gives better performance than IMPORT. This is true no matter whether you are using the INSERT or REPLACE modes of LOAD.
- When appending a substantial amount of data to an existing table with indexes (using IMPORT INSERT, or LOAD INSERT), LOAD gives slightly better performance than IMPORT.
- When appending a small amount of data to an existing large table with indexes (using IMPORT INSERT, or LOAD INSERT), IMPORT may perform better than LOAD since IMPORT will not incur the cost of rebuilding the entire index.
- If you are using the IMPORT 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.

Defining Referential Constraints: Referential integrity is imposed by adding referential constraints to table and column definitions. Referential constraints are established with the FOREIGN KEY Clause, and the REFERENCES Clause in the CREATE TABLE or ALTER TABLE statements. A referential constraint cannot be associated with a typed table.

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 "Modifying a Table in Both Structure and Content" on page 216.

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 218). 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 48.

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 INTEGRITY 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 check constraint can be associated with a typed table.

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 “Modifying a Table in Both Structure and Content” on page 216.

Creating a Typed Table

You can create a typed table using a variant of the CREATE TABLE statement. You can also create a hierarchy of typed tables that is based on a hierarchy of structured types. The following example illustrates creation of a table hierarchy based on the type hierarchy described in “Creating a User-Defined Structured Type” on page 180:

```
CREATE TABLE Department OF Department_t
   (REF IS Oid USER GENERATED);

CREATE TABLE Person OF Person_t
   (REF IS Oid USER GENERATED);
CREATE TABLE Employee OF Employee_t UNDER Person
   INHERIT SELECT PRIVILEGES
   (Dept WITH OPTIONS SCOPE Department);
CREATE TABLE Student OF Student_t UNDER Person
   INHERIT SELECT PRIVILEGES;
CREATE TABLE Manager OF Manager_t UNDER Employee
   INHERIT SELECT PRIVILEGES;
CREATE TABLE Architect OF Architect_t UNDER Employee
   INHERIT SELECT PRIVILEGES;
```

The first typed table created above is `Department`. This table is defined to be OF type `Department_t`, so it will hold instances of that type. This means that it will have a column corresponding to each attribute of the structured type `Department_t`. Because typed tables contain objects that can be referenced by other objects, every typed table must have an “object identifier” (OID) column as its first column. In this example, the type of the OID column will be REF(`Department_t`), and its column name (`Oid`) is given in the REF IS...USER GENERATED clause. The USER GENERATED part of this clause indicates

that the initial value for the OID column of each newly inserted row will be provided by the user when inserting a row; once inserted, the OID column cannot be updated.

The next typed table above, Person, is of type Person_t. The type Person_t is the root of a type hierarchy, so we need to create a corresponding "table hierarchy" if we want to store instances of type Person_t and its subtypes. Thus, after creating the table Person, we create two "subtables" of the Person table, Employee and Student, and also two subtables of the Employee table, Manager and Architect. Just as a subtype inherits the attributes of its supertype, a subtable inherits the columns of its supertable — including the OID column. (Note: A subtable must reside in the same schema as its supertable.) Rows in the Employee subtable, for example, will therefore have a total of six columns: Oid, Name, Age, SerialNum, Salary, and Dept.

The INHERIT SELECT PRIVILEGES clause specifies that the subtable being defined, such as Employee, should (at least initially) be readable by the same users and groups as the "supertable", such as Person, UNDER which it is created. Any user or group holding a SELECT privilege on the supertable will be granted SELECT privilege on the newly created subtable, with the subtable definer being the grantor of this privilege.

Note: Privileges may be granted and revoked independently at every level of a table hierarchy. Thus, the inherited SELECT privileges on a subtable may be revoked after the subtable has been created if the definer of the subtable does not wish for them to remain granted. While doing so does not prevent a user with SELECT privilege on the supertable from seeing those columns of the subtable's rows, it does prevent them from seeing the additional columns that appear only at the level of the subtable because a user can only operate directly on a subtable if they hold the necessary privilege on that subtable.

The WITH OPTIONS SCOPE clause in the CREATE statement for the Employee table declares that the Dept column of this table has a "scope" of Department. This means that the reference values in this column of the Employee table are intended to refer to objects in the Department table. The scope information is needed if the user wants to be able to dereference these references in SQL statements using the new SQL dereference operator (–>).

This example has shown how a table hierarchy can be defined, based on a corresponding hierarchy of structured types, in order to create a database in which objects of particular types and subtypes can be stored and managed. Every table hierarchy has a "root table", which has an OID column plus a column for each attribute of its declared type. In addition, it can have a number of "subtables", each of which is created UNDER the root table or

some other appropriate “supertable” within the table hierarchy. This example has also shown how scopes are specified for reference attributes.

A SELECT, UPDATE, or DELETE statement that operates on a supertable automatically operates on all its subtables as well. For example, an UPDATE statement on the `Employee` table might affect rows in `Employee`, `Manager`, and `Architect` tables, but an UPDATE statement on the `Manager` table can only affect `Manager` rows.

See *SQL Reference* for more information on the CREATE TABLE statement (or the CREATE VIEW statement) and how to establish subtype/supertype relationships between typed tables. (For an introduction to CREATE VIEW you could see “Creating a Typed View” on page 184.)

Populating a Typed Table

After creating the structured types and then creating the corresponding tables and subtables, you will have a database like the following:

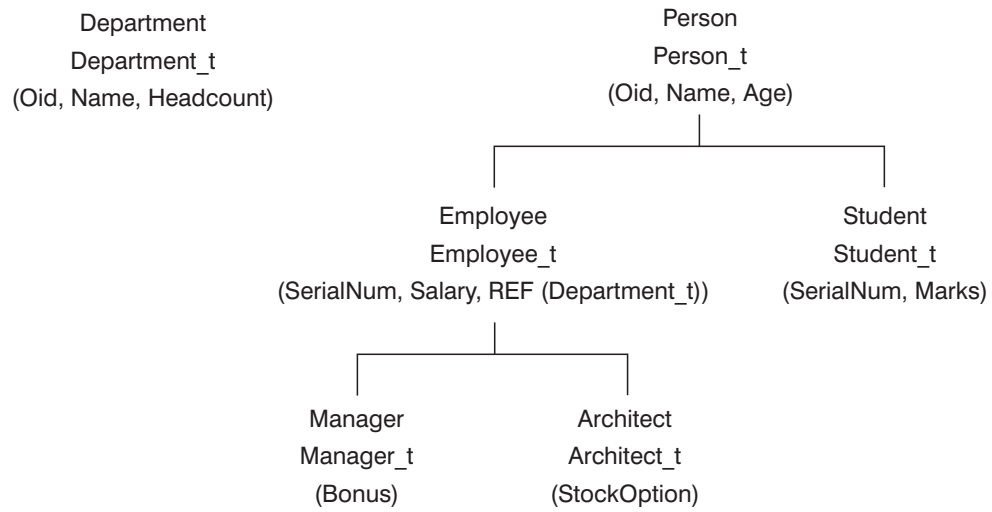


Figure 20.

Once the hierarchy is established, you will need to populate the tables with data. This may be done as shown in the following example:

```
INSERT INTO Department (Oid, Name, Headcount)
   VALUES(Department_t('1'), 'Toy', 15);
INSERT INTO Department (Oid, Name, Headcount)
   VALUES(Department_t('2'), 'Shoe', 10);

INSERT INTO Person (Oid, Name, Age)
   VALUES(Person_t('a'), 'Andrew', 20);
```

```
INSERT INTO Person (Oid, Name, Age)
   VALUES(Person_t('b'), 'Bob', 30);
INSERT INTO Person (Oid, Name, Age)
   VALUES(Person_t('c'), 'Cathy', 25);
INSERT INTO Employee (Oid, Name, Age, SerialNum, Salary, Dept)
   VALUES(Employee_t('d'), 'Dennis', 26, 105, 30000, Department_t('1'));
INSERT INTO Employee (Oid, Name, Age, SerialNum, Salary, Dept)
   VALUES(Employee_t('e'), 'Eva', 31, 83, 45000, Department_t('2'));
INSERT INTO Employee (Oid, Name, Age, SerialNum, Salary, Dept)
   VALUES(Employee_t('f'), 'Franky', 28, 214, 39000, Department_t('2'));
INSERT INTO Student (Oid, Name, Age, SerialNum, Marks)
   VALUES(Student_t('g'), 'Gordon', 19, 10245, 90);
INSERT INTO Student (Oid, Name, Age, SerialNum, Marks)
   VALUES(Student_t('h'), 'Helen', 20, 10357, 70);
INSERT INTO Manager (Oid, Name, Age, SerialNum, Salary, Dept, Bonus)
   VALUES(Manager_t('i'), 'Iris', 35, 251, 55000, Department_t('1'), 12000);
INSERT INTO Manager (Oid, Name, Age, SerialNum, Salary, Dept, Bonus)
   VALUES(Manager_t('j'), 'Christina', 10, 317, 85000, Department_t('1'), 25000);
INSERT INTO Manager (Oid, Name, Age, SerialNum, Salary, Dept, Bonus)
   VALUES(Manager_t('k'), 'Ken', 55, 482, 105000, Department_t('2'), 48000);
INSERT INTO Architect (Oid, Name, Age, SerialNum, Salary, Dept, StockOption)
   VALUES(Architect_t('l'), 'Leo', 35, 661, 92000, Department_t('2'), 20000);
INSERT INTO Architect (Oid, Name, Age, SerialNum, Salary, Dept, StockOption)
   VALUES(Architect_t('m'), 'Brian', 7, 882, 112000,
          (SELECT Oid FROM Department WHERE name = 'Toy'), 30000);
```

Notice from the example that first value in each inserted row is the OID for the data being inserted into the tables. Also, when inserting data into a subtable, note that data must be provided for its inherited columns. Finally, notice that any reference-valued expression of the appropriate type can be used to initialize a reference attribute. In most cases above, the Dept reference of the employees is input as an appropriately type-casted constant; however, in the case of Brian, the reference is obtained using a subquery.

Following the above INSERT statements, we can now query the typed tables. For example, here is the result we would obtain if we now ask DB2 to ″SELECT Name, Age FROM Person″, which prints the names and ages of all persons (in Person or its subtables) in our database:

```
NAME                 AGE
-------------------- -----------
Andrew                        20
Bob                           30
Dennis                        26
Eva                           31
Franky                        28
Gordon                        19
Helen                         20
Iris                          35
Christina                     10
Ken                           55
```

```
Leo                             35
Brian                            7

  12 record(s) selected.
```

Similarly, here is the result of the query ″SELECT Name, Salary, Dept–>Name FROM Employee″, which prints the names, salaries, and department names of all the employees in the database:

```
NAME                 SALARY      NAME
-------------------- ----------- --------------------
Dennis                     30000 Toy
Eva                        45000 Shoe
Franky                     39000 Shoe
Iris                       55000 Toy
Christina                  85000 Toy
Ken                       105000 Shoe
Leo                        92000 Shoe
Brian                     112000 Toy

  8 record(s) selected.
```

Note: In the second SELECT statement above, the dereference operator (–>) is used. The dereference operator returns the named column value from the target table of a scoped reference. In the expression “Dept —> Name”, “Dept” is a reference column whose scope (target table) is “Department”, and “Name” is the name of a column in that target table.

Hierarchy Table

A hierarchy table is a table that is associated with the implementation of a typed table hierarchy. It is created at the same time as the root table of the hierarchy. When creating a root table, an optional HIERARCHY clause can be used to specify the name of the hierarchy table that is associated with the root table. If you do not specify a name, the name of the hierarchy table is the same as the name of the root table, followed by a system-generated unique suffix. A hierarchy table cannot be directly referenced in an SQL statement.

The hierarchy table contains one column for each unique column in the hierarchy. Give the hierarchy show in Figure 20 on page 169, the hierarchy table would contain these columns: `Oid`, `Name`, `Age`, `SerialNum`, `Salary`, `Marks`, `Bonus`, and `StockOption`. (The actual order of the columns depends on the order that the tables were created.) In addition, there is an extra column for the type ID so that the DB2 database manager can tell the type of a given row.

Suppose the type IDs are as follows: `Person` uses 10, `Employee` uses 25, `Manager` uses 35, `Architect` uses 45, and `Student` uses 100. Then given the

values inserted into the hierarchy as shown following Figure 20 on page 169, the populated hierarchy table would appear as follows:

Table 21. Hierarchy Table

(type)	Oid	Name	Age	SerialNum	Salary	Marks	Bonus	StockOption
10	a	Andrew	20	—	—	—	—	—
10	b	Bob	30	—	—	—	—	—
10	c	Cathy	25	—	—	—	—	—
25	d	Dennis	26	105	30000	—	—	—
25	e	Eva	31	83	45000	—	—	—
25	f	Franky	28	214	39000	—	—	—
100	g	Gordon	19	10245	—	90	—	—
100	h	Helen	20	10357	—	70	—	—
35	i	Iris	35	251	55000	—	12000	—
35	j	Christina	10	371	85000	—	25000	—
35	k	Ken	55	482	105000	—	48000	—
45	l	Leo	35	661	92000	—	—	20000
45	m	Brian	7	882	112000	—	—	30000

Notice that when a column does not apply to a given row, then the value is NULL (as shown by the "—").

The SQL optimizer uses the hierarchy table to generate access plans for processing queries written against the individual tables in the hierarchy.

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 85 for additional considerations on the use of multiple DMS table spaces for a single table.

Refer to 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 73.

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. (This assumes a 4 KB page size for the table space.) The size of the table can be as large as 64 GB (or the available disk space) times the number of database partitions. If the page size for the table space was 8 KB, the size of the table can be as large as 128 GB (or the available disk space) times the number of database partitions. If the page size for the table space was 16 KB, the size of the table can be as large as 256 GB (or the available disk space) times the number of database partitions. If the page size for the table space was 32 KB, the size of the table can be as large as 512 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 cannot use triggers with nicknames.

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. The trigger can be activated before or after the INSERT, UPDATE, or DELETE statement to which it refers. Refer to the *SQL Reference* for complete syntax information on the CREATE TRIGGER statement. Refer to the *Application Development Guide* for information about creating and using triggers.

Trigger Dependencies

All dependencies of a trigger on some other object are recorded in the SYSCAT.TRIGDEP catalog. A trigger can depend on many objects. These objects and the dependent trigger are presented in detail in the *SQL Reference* discussion on the DROP statement.

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 233.)

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, refer to "Updating Statistics for User-Defined Functions" in *Administration Guide, Performance.*

For details on using the CREATE FUNCTION statement to write a UDF to suit your specific application, refer to the *Application Development Guide.* Refer to the *SQL Reference* for details on UDF syntax.

Creating a Function Mapping

In a federated database, create a function mapping when you need to map a local function or a local function template (described in "Creating a Function Template" on page 178) with a function at one or more data sources. Default function mappings are provided for many data source functions.

Function mappings are useful when:

- New, built-in functions become available at a data source.
- You need to map a user-defined function at a data source to a local function.
- An application requires different default behavior than that provided by the default mapping.

Function mappings defined with CREATE FUNCTION MAPPING statements are stored in the federated database.

Functions (or function templates) must have the same number of input parameters as the data source function. Additionally, the data types of the input parameters on the federated side should be compatible with the data types of the input parameters on the data source side.

Use the CREATE FUNCTION MAPPING statement to create a function mapping. For example, to create a function mapping between an Oracle AVGNEW function and a DB2 equivalent at server ORACLE1:

```
CREATE FUNCTION MAPPING ORAVGNEW FOR SYSIBM.AVG(INT) SERVER ORACLE1
OPTIONS (REMOTE_NAME 'AVGNEW')
```

You must hold one of the SYSADM or DBADM authorities at the federated database to use this statement. Function mapping attributes are stored in SYSCAT.FUNCMAPPINGS.

The federated server will not bind input host variables or retrieve results of LOB, LONG VARCHAR/VARGRAPHIC, DATALINK, distinct and structured types. No function mapping can be created when an input parameter or the returned value includes one of these types.

For additional details on using and creating function mappings, refer to the *Application Development Guide*. Refer to the *SQL Reference* for details on CREATE FUNCTION MAPPING syntax.

Creating a Function Template

In a federated system, function templates provide “anchors” for function mappings. They are used to enable the mapping of a data source function when a corresponding DB2 function does not exist at the federated server. A function mapping requires the presence of a function template or an existing similar function at DB2.

The template is just a function shell: name, input parameters, and the return value. There is no local executable for the function.

Because there is no local executable for the function, it is possible that a call to the function template will fail even though the function is available at the data source. For example, consider the query:

```
SELECT myfunc(C1)
FROM nick1
WHERE C2 < 'A'
```

If DB2 and the data source containing the object referenced by nick1 do not have the same collating sequence, the query will fail because the comparison must be done at DB2 while the function is at the data source. If the collating sequences were the same, the comparison operation could be done at the data source that has the underlying function referenced by myfunc.

Functions (or function templates) must have the same number of input parameters as the data source function. The data types of the input parameters on the federated side should be compatible with the data types of the input parameters on the data source side. These requirements apply to returned values as well.

You create function templates using the CREATE FUNCTION statement with the AS TEMPLATE keyword. After the template is created, you map the template to the data source using the CREATE FUNCTION MAPPING statement.

For example, to create a function template and a function mapping for function MYS1FUNC on server S1:

```
CREATE FUNCTION MYFUNC(INT) RETURNS INT AS TEMPLATE

CREATE FUNCTION MAPPING S1_MYFUNC FOR MYFUNC(INT) SERVER S1 OPTIONS
(REMOTE_NAME 'MYS1FUNC')
```

For details on using and creating function templates, refer to the *Application Development Guide*. Refer to the *SQL Reference* for details on CREATE FUNCTION syntax.

Creating a User-Defined Type (UDT)

A user-defined type (UDT) is a named data type that is created in the database by the user. A UDT can be a distinct type which shares a common representation with a built-in data type or a structured type which has a sequence of named attributes that each have a type. A structured type can be a subtype of another structured type (called a supertype), defining a type hierarchy.

UDTs support strong typing, which means that even though they share the same representation as other types, values of a given UDT are considered to be compatible only with values of the same UDT or UDTs in the same type hierarchy.

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, refer to the *SQL Reference*.

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.

You can drop a UDT only if:

- It is **not** used in a column definition for an existing table.
- It is **not** used as the type of an existing typed table or typed view.
- 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 User-Defined Distinct Type

A user-defined distinct type is a data type derived from an existing type, such as an integer, decimal, or character type. You can create a distinct type by using the CREATE DISTINCT TYPE statement.

The following SQL statement creates the distinct type `t_educ` as a smallint:

```
CREATE DISTINCT TYPE T_EDUC AS SMALLINT WITH COMPARISONS
```

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 (as in the example). The WITH COMPARISONS clause cannot be specified if the source data type is a large object, a DATALINK, LONG VARCHAR, or LONG VARGRAPHIC type.

Instances of distinct types 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 distinct type.

After you have created a distinct type, 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
```

Creating the distinct type also generates support to cast between the distinct type and the source type. Hence, a value of type T_EDUC can be cast to a SMALLINT value and SMALLINT value can be cast to a T_EDUC value.

Refer to the *SQL Reference* for complete syntax information on the CREATE DISTINCT TYPE statement. Refer to the *Application Development Guide* for information about creating and using a distinct type.

You can transform UDTs into base data types, and base data types into UDTs, using transformations. Creation of a transform function is through a CREATE TRANSFORM statement.

Support for transforms is also found through the CREATE METHOD statement and extensions to the CREATE FUNCTION statement. Refer to the *SQL Reference* for details on this support.

Creating a User-Defined Structured Type

A structured type is a user-defined type that contains one or more attributes, each of which has a name and a data type of its own. A structured type can serve as the type of a table, in which each column of the table derives its name and data type from one of the attributes of the structured type. A structured type may be created as a subtype of another structured type, called

its “supertype”. In this case, the subtype inherits all the attributes of the supertype, and may optionally add additional attributes of its own.

For example, consider the following user-defined structured types:

```
CREATE TYPE Department_t AS (Name VARCHAR(20), Headcount INT)
   MODE DB2SQL;
CREATE TYPE Person_t AS (Name VARCHAR(20), Age INT)
   MODE DB2SQL;
CREATE TYPE Employee_t UNDER Person_t
   AS (SerialNum INT, Salary INT, Dept REF(Department_t))
   MODE DB2SQL;
CREATE TYPE Student_t UNDER Person_t AS (SerialNum INT, Marks INT)
   MODE DB2SQL;
CREATE TYPE Manager_t UNDER Employee_t AS (Bonus INT)
   MODE DB2SQL;
CREATE TYPE Architect_t UNDER Employee_t AS (StockOption INT)
   MODE DB2SQL;
```

The AS clause provides the attribute definitions associated with the type.

The MODE DB2SQL clause is used to specify the mode of the type. DB2SQL is the only value for mode currently supported.

The UNDER clause specifies that the structured type is being defined as a subtype of the specified supertype.

The first structured type above (Department_t) is a type with two attributes: Name and Headcount. The second structured type (Person_t) is another type with two attributes: Name and Age. The type Person_t has two subtypes, Employee_t and Student_t, that each inherit the attributes of Person_t and also have several additional attributes that are specific to their particular types. Note that the Dept attribute of Employee_t is a reference, of type REF(Department_t), that can refer to an object of type Department_t. Finally, Manager_t and Architect_t are both subtypes of Employee_t; they inherit all the attributes of Employee_t and extend them further as appropriate for their types. Thus, an instance of type Manager_t will have a total of six attributes: Name, Age, SerialNum, Salary, Dept, and Bonus.

This example showing user-defined structured types contains definitions for two “type hierarchies”. One is the Department_t type hierarchy, which consists only of the type Department_t (and therefore isn’t much of a hierarchy). The other is the Person_t type hierarchy, which consists of the type Person_t, two subtypes of Person_t, namely Employee_t and Student_t, and two subtypes of Employee_t, namely Manager_t and Architect_t. The Department_t type and Person_t type are “root types” since they are not subtypes of any other type (that is, neither one has an UNDER clause in its type definition).

Refer to *SQL Reference* for more information on the CREATE TYPE (Structured) statement.

Creating a Type Mapping

In a federated system, a type mapping lets you map specific data types in data source tables and views to DB2 distinct data types. A type mapping can apply to one data source or a range (type, version) of data sources.

Default data type mappings are provided for built-in data source types and built-in DB2 types. New data type mappings (that you create) will be listed in the SYSCAT.TYPEMAPPINGS view.

You create type mappings with the CREATE TYPE MAPPING statement. You must hold one of the SYSADM or DBADM authorities at the federated database to use this statement.

An example of a type mapping statement is:

```
CREATE TYPE MAPPING MY_ORACLE_DEC FROM SYSIBM.DECIMAL(10,2)
TO SERVER ORACLE1 TYPE NUMBER([10..38],2)
```

You cannot create a type mapping for a LOB, LONG VARCHAR/VARGRAPHIC, DATALINK, structured or distinct type.

For details on using and creating type mappings, refer to the *Application Development Guide*. Refer to the *SQL Reference* for details on CREATE TYPE MAPPING syntax.

Creating a View

Views are derived from one or more base tables, nicknames, 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, refer to 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
```

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. This can happen by creating a view based on an underlying table. Applications that use the underlying table are not affected by the creation of the new view. New applications can use the created view for different purposes than those applications that use the underlying table.
- Sum the values in a column, select the maximum values, or average the values.
- Provide access to information in one or more data sources. You can reference nicknames within the CREATE VIEW statement and create multi-location / global views (the view could join information in multiple data sources located on different systems).

  When you create a view that references nicknames using standard CREATE VIEW syntax, you will see a warning alerting you to the fact that the authentication ID of view users will be used to access the underlying object or objects at data sources instead of the view creator authentication ID. Use the FEDERATED keyword to suppress this warning.

An alternative to creating a view is to use a nested or common table expression to reduce catalog lookup and improve performance. Refer to the *SQL Reference* for more information about common table expressions.

Creating a Typed View

You can create a typed view using the CREATE VIEW statement. For example, to create a view of the typed `Department` table that we created earlier, we can define a structured type that has the desired attributes and then create a typed view using that type:

```
CREATE TYPE VDepartment_t AS (Name VARCHAR(20))
   MODE DB2SQL;
```

```
CREATE VIEW VDepartment OF VDepartment_t MODE DB2SQL
   (REF IS VOid USER GENERATED)
   AS SELECT VDepartment_t(Varchar(Oid)), Name FROM Department;
```

The OF clause in the CREATE VIEW statement tells the system that the columns of the view are to be based on the attributes of the indicated structured type (in this case VDepartment_t).

The MODE DB2SQL clause specifies the mode of the typed view. This is the only valid mode currently supported.

The REF IS... clause is identical to that of the typed CREATE TABLE statement. It provides a name for the view's OID column (VOid in this case), which is the first column of the view. Typed views, like typed tables, require an OID column to be specified (in the case of a root view) or inherited (in the case of a subview, as will be shown shortly).

The USER GENERATED clause specifies that the initial value for the OID column must be provided by the user when inserting a row. Once inserted, the OID column cannot be updated.

The "body" of the view, which follows the keyword AS, is a SELECT statement that determines the content of the view. The column-types returned by this SELECT statement must be compatible with the column-types of the typed view, including the initial object ID column.

To illustrate the creation of a typed view hierarchy, the following example defines a view hierarchy that omits some sensitive data and eliminates some type distinctions from the Person table hierarchy created earlier under "Creating a Typed Table" on page 167:

```
CREATE TYPE VPerson_t AS (Name VARCHAR(20))
   MODE DB2SQL;

CREATE TYPE VEmployee_t UNDER VPerson_t
   AS (Salary INT, Dept REF(VDepartment_t))
   MODE DB2SQL;

CREATE VIEW VPerson OF VPerson_t MODE DB2SQL
   (REF IS VOid USER GENERATED)
   AS SELECT VPerson_t (Varchar(Oid)), Name FROM ONLY(Person);

CREATE VIEW VEmployee OF VEmployee_t MODE DB2SQL
   UNDER VPerson INHERIT SELECT PRIVILEGES
   (Dept WITH OPTIONS SCOPE VDepartment)
   AS SELECT VEmployee_t(Varchar(Oid)), Name, Salary,
      VDepartment_t(Varchar(Dept))
   FROM Employee;
```

The two CREATE TYPE statements create the structured types that are needed to create the object view hierarchy for this example.

The first typed CREATE VIEW statement above creates the root view of the hierarchy, VPerson, and is very similar to the VDepartment view definition. The difference is the use of ONLY(Person) to ensure that only the rows in the Person table hierarchy that are in the Person table (and not in any subtable) are included in the VPerson view. This ensures that the Oid values in VPerson are unique compared with the Oid values in VEmployee. The second CREATE VIEW statement creates a subview VEmployee under the view VPerson. As was the case for the UNDER clause in the CREATE TABLE...UNDER statement, the UNDER clause when creating a view establishes the superview/subview relationship. (Note: The subview must be created in the same schema as its superview.) As was the case for typed tables, columns are inherited by subviews. Rows in the VEmployee view will inherit the columns VOid and Name from VPerson and have the additional columns Salary and Dept associated with the type VEmployee_t.

The INHERIT SELECT PRIVILEGES clause has the same meaning here as in the typed CREATE TABLE statement.

Similarly, the WITH OPTIONS clause in a typed view definition plays the same role as it does in a typed table definition — it allows column options such as SCOPE to be specified. As well, a new column option, READ ONLY (not used in our example), is provided for columns of typed views. This clause is used to force a superview column to be marked as read-only so that a later subview definition can legitimately specify an expression for the same column that is implicitly read-only.

If a view has a reference column (like VEmployee's Dept column), a scope must be associated with the column if it is to be usable in SQL dereference operations. If no scope is specified for the reference column of the view and the underlying table or view column was scoped, then the underlying column's scope is passed on to the view's reference column. It can be explicitly given a scope by using WITH OPTIONS, as in our example where the Dept column of the VEmployee view gets the VDepartment view as its scope. The column would remain unscoped if the underlying table or view column did not have a scope and none was explicitly assigned in the view definition (or later by using the ALTER VIEW statement).

There are several important rules associated with restrictions on the queries for typed views found in the *SQL Reference* that you should read carefully before attempting to create and use a typed view.

Creating a Summary Table

A *summary table* is a table whose definition is based on the result of a query. As such, the summary table typically contains pre-computed results based on the data existing in the table or tables that its definition is based on. If the SQL compiler determines that a query will run more efficiently against a summary table than the base table, the query executes against the summary table, and you obtain the result faster than you otherwise would.

The creation of a summary table with the replication option can be used to replicate tables across all nodes in a partitioned database environment. These are known as "replicated summary tables". See "Replicated Summary Tables" on page 74 for more information.

Note: Summary tables are not used with static SQL or nicknames.

In general a summary table, or a replicated summary table, is used for optimization of a query if the isolation level of the summary table, or the replicated summary table, is higher than or equal to the isolation level of the query. For example, if a query is running under the cursor stability (CS) isolation level, only summary tables, and replicated summary tables, that are defined under CS or higher isolation levels are used for optimization.

To create a summary table, you use the CREATE SUMMARY TABLE statement with the AS *fullselect* clause and the IMMEDIATE or REFRESH DEFERRED options. When you create the summary table, you have the option of specifying whether the summary table is refreshed automatically when the base table is changed, or whether it is refreshed by using the REFRESH TABLE statement. To have the summary table refreshed automatically when changes are made to the base table or tables, specify the REFRESH IMMEDIATE keyword. An immediate refresh is useful when:

- You have queries that take a long time to complete when run against a base table
- The base table or tables are infrequently changed
- The refresh is not expensive.

The summary table, in this situation, can provide pre-computed results. If you want the refresh of the summary table to be deferred, specify the REFRESH DEFERRED keyword. Summary tables specified with REFRESH DEFERRED will **not** reflect changes to the underlying base tables. You should use summary tables where this is not a requirement. For example, if you run DSS queries, you would use the summary table to contain legacy data.

A summary table defined with REFRESH DEFERRED may be used in place of a query when it:

- Conforms to the restrictions for a fullselect of a refresh immediate summary table, except:
  - The SELECT list is not required to include COUNT(*) or COUNT_BIG(*)
  - The SELECT list can include MAX and MIN column functions
  - A HAVING clause is allowed.

The SQL special register CURRENT REFRESH AGE SQL is set to ANY or has a value of 99999999999999. The collection of nines is the maximum value allowed in this special register which is a timestamp duration value with a data type of DECIMAL(20,6).

Note: Summary tables defined with REFRESH DEFERRED are not used to optimize static SQL.

You use the CURRENT REFRESH AGE special register to specify the amount of time that the summary table with deferred refresh can be used for a dynamic query before it must be refreshed. To set the value of the CURRENT REFRESH AGE special register, you can use the SET CURRENT REFRESH AGE statement. For more information about the CURRENT REFRESH AGE special register and the SET CURRENT REFRESH AGE statement, refer to the *SQL Reference*.

Summary tables defined with REFRESH IMMEDIATE are applicable to both static and dynamic queries and do not need to use the CURRENT REFRESH AGE special register.

Note: Setting the CURRENT REFRESH AGE special register to a value other than zero should be done with caution. By allowing a summary table that may not represent the values of the underlying base table to be used to optimize the processing of the query, the result of the query may *not* accurately represent the data in the underlying table. This may be reasonable when you know the underlying data has not changed, or you are willing to accept the degree of error in the results based on your knowledge of the data.

With activity affecting the source data, a summary table over time will no longer contain accurate data. You will need to use the REFRESH TABLE statement. Refer to the *SQL Reference* for more information.

If you want to create a new base table that is based on any valid *fullselect*, specify the DEFINITION ONLY keyword when you create the table. When the create table operation completes, the new table is not treated as a summary table, but rather as a base table. For example, you can create the exception tables used in LOAD and SET INTEGRITY as follows:

```
CREATE TABLE XT AS
(SELECT T.*, CURRENT TIMESTAMP AS TIMESTAMP,CLOB(",32K)
AS MSG FROM T) DEFINITION ONLY
```

Here are some of the key restrictions regarding summary tables:

1. You cannot alter a summary table.
2. You cannot alter the length of a column for a base table if that table has a summary table.
3. You cannot import data into a summary table.
4. You cannot create a unique index on a summary table.
5. You cannot create a summary table based on the result of a query that references one or more nicknames.

Refer to the *SQL Reference* for a complete statement of summary table restrictions.

Creating an Alias

An alias is an indirect method of referencing a table, nickname, 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 a Wrapper

In a federated database, the CREATE WRAPPER statement registers a wrapper. The statement defines the mechanism by which a federated server can interact with a certain category of data source.

Specific libraries must be used for specific data source types, versions, communication protocols, and operating systems. For example, AS/400 and DB2 for OS/390 data sources are accessed using the ″libdrda.dll″ library for federated databases operating on Windows NT operating systems using APPC communications.

Wrappers can be created in the Control Center or from the command line processor. In both cases, creating a wrapper registers it to the federated database.

The following SQL statement registers the wrapper ORACLE8 on a Windows NT operating system:

```
CREATE WRAPPER ORACLE8 LIBRARY 'libnet8.dll'
```

You must have SYSADM or DBADM authority at the federated database to use this statement.

For details on using the CREATE WRAPPER statement, refer to the *Application Development Guide*. Refer to the *SQL Reference* for details on syntax.

Creating a Server

In a federated database, create servers to define data sources to DB2 and describe their characteristics: name, wrapper, type, version, location, and options. This information is used to map nicknames to specific data management systems and to provide information to the DB2 optimizer. Server information is located in the SYSCAT.SERVERS and SYSCAT.SERVEROPTIONS catalog views.

Note: In this section, servers represent data sources, not DRDA servers or DB2 DBMSs.

You can create servers from the Control Center or the command line processor.

The following sample SQL statement creates the Oracle server ORA8:

```
CREATE SERVER ORA8 TYPE ORACLE VERSION 8 WRAPPER ORACLE8 OPTIONS
(NODE 'ONODE')
```

The following sample SQL statement creates the DB2 server DB2TEST:

```
CREATE SERVER DB2TEST TYPE DB2 VERSION 6.1 WRAPPER DB2UDB OPTIONS
(NODE 'DB2TEST', DBNAME 'TEST1')
```

The definition of NODE, in SERVER SQL statements, varies depending on the data source. If the data source is a DB2 DBMS, the value refers to an instance of DB2 that has one or more databases. In the previous example, note that the DBNAME option specifies the database name. If the data source is a DB2 for OS/390 DBMS, the value refers to a specific node for a subsystem. If the data source is an Oracle DBMS, the value refers to the actual database (the DBNAME option is not needed).

You must have SYSADM or DBADM authority at the federated database to use this statement.

For additional details on using the CREATE SERVER statement, refer to the *SQL Reference*.

You can create user mappings to manage differences in authentication processing between DB2 and data source servers. User mappings are discussed in detail in "User Mappings" on page 301.

When a server is dropped, all objects dependent on that server are dropped (user mappings, nicknames, function mappings, type mappings, plans, etc.).

Provide server options when creating a server. These options provide necessary details about the server (such as the node name). Server options can also set specific performance and security values.

Using Server Options to Help Define Data Sources and Facilitate Authentication Processing

You can set variables called *server options* to values that affect how a federated server accesses data sources. This section:

- Explains the purpose of server options
- Describes what SQL statements you use to specify server options
- Shows the server options and their settings

Purposes of Server Options: In general, you use server options to:

- Supply and update information about data sources. A server reference includes both basic information about a data source—for example, its name—and information that can change over time. Some of the changeable information is conveyed by values assigned to server options. For example, the value assigned to the cpu_ratio option indicates whether the data source's CPU is faster or slower than the DB2 system CPU. If the DB2 system gets one or more processor upgrades, this value should change.
- Facilitate authentication. You can set some server options to ensure that user IDs and passwords are sent to the data source in the proper case. For example, you can set the fold_id option so that before the federated server sends a user ID to a data source, the federated server transforms the name to the case (upper or lower) that the data source requires. Alternatively, if you define the user ID to the federated server in the required case, you can set the fold_id option to prevent the server from trying to change the case and consuming overhead in the process.
- Optimize queries. Some server options and their values facilitate optimization. To illustrate: in the CREATE SERVER statement, you can specify certain performance statistics as option values. For example, you can set the cpu_ratio option to a value that indicates the relative speeds of the data source's and federated server's CPUs. And you can set the io_ratio option to a value that indicates the relative rates of the data source's and federated server's I/O devices. When you run CREATE SERVER, these statistics are added to the catalog view SYSCAT.SERVEROPTIONS, and the optimizer uses them in developing its access plan for the data source. If a

statistic changes (as might happen, for instance, if the data source CPU is upgraded), you can use the ALTER SERVER statement to update SYSCAT.SERVEROPTIONS with this change. The optimizer then uses your update in developing its next access plan for the data source.

SQL for Server Options: There are three SQL statements in which you can assign values to server options: CREATE SERVER, ALTER SERVER, and SET SERVER OPTION.

Use the CREATE SERVER statement to set an option to a value that persists indefinitely over time for multiple connections to a data source. With this statement, you can set an option to a value other than the default or, if an option has no default value, you can set it to an initial value.

Use the ALTER SERVER statement if, after setting a server option to a value with the CREATE SERVER statement, you want to set it to a different value that persists over multiple connections.

Use the SET SERVER OPTION statement to change server option values temporarily for the duration of a single connection to a database. SET SERVER OPTION statements must be issued first within the first unit of work following the connection to the data source.

For example, to temporarily enable the use of plan hints for the Oracle server ORASEB1, issue the statement:

```
SET SERVER OPTION plan_hints TO 'Y' FOR SERVER ORASEB1
```

Server Options and Their Settings: The table below describes the server options and the values that you can set them to. Unless otherwise stated, all server option values must be enclosed in single quotes.

Table 22. Server Options and Their Settings

Option	Valid Settings	Default Setting
collating_sequence	Specifies whether the data source uses the same default collating sequence as the federated database, based on the code set and the country information. If a data source has a collating sequence that differs from DB2's collating sequence, most operations depending on DB2's collating sequence cannot be remotely evaluated at a data source. An example is executing MAX column functions against a nickname character column at a data source with a different collating sequence. Because results might differ if the MAX function is evaluated at the remote data source, DB2 will perform the aggregate operation and the MAX function locally. If your query contains an equal sign, it is possible to push-down that portion of the query even if the collating sequences are different (set to 'N'). For example, the predicate C1 = 'A' could be pushed-down to a data source. Of course, such queries cannot be pushed-down when the collating sequence at the data source is case-insensitive. When a data source is case-insensitive, the results from C1= 'A' and C1 = 'a' are the same, which is not acceptable in a case-sensitive environment (DB2). Administrators can create federated databases with a particular collating sequence that matches the data source collating sequence. This approach may speed performance if all data sources use the same collating sequence or if most or all column functions are directed against data sources that use the same collating sequence. 'Y' Data source's collating sequence is the same as federated database's. 'N' Data source's collating sequence is not the same as federated database's. 'I' Data source's collating sequence is different from federated database's and is case-insensitive (for example, 'TOLLESON' and 'TolLESon' are considered equal).	'N'
comm_rate	Specifies the communication rate between a federated server and its associated data sources. Expressed in megabytes per second.	'2.0'

Table 22. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
connectstring	Specifies initialization properties needed to connect to an OLE DB provider. For the complete syntax and semantics of the connection string, see the ″Data Link API of the OLE DB Core Components″ in the *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK, Microsoft Press, 1998*.	None
cpu_ratio	Indicates how much faster or slower a data source's CPU runs than the federated server's CPU.	'1.0'
dbname	Name of the data source database that you want the federated server to access. Required for DB2 family data sources; does not apply to Oracle** data sources.	None.
fold_id (See notes 1 and 4 at the end of this table.)	Applies to user IDs that the federated server sends to data sources for authentication. Valid values are: 'U' The federated server folds the user ID to uppercase before sending it to the data source. This is a logical choice for DB2 Family and Oracle** data sources (See note 2 at end of this table.) 'N' The federated server does nothing to the user ID before sending it to the data source. (See note 2 at end of this table.) 'L' The federated server folds the user ID to lowercase before sending it to the data source. If none of these settings are used, the federated server tries to send the user ID to the data source in uppercase. If the user ID fails, the server tries sending it in lowercase.	None.
fold_pw (See notes 1, 3 and 4 at the end of this table.)	Applies to passwords that the federated server sends to data sources for authentication. Valid values are: 'U' The federated server folds the password to uppercase before sending it to the data source. This is a logical choice for DB2 Family and Oracle** data sources. 'N' The federated server does nothing to the password before sending it to the data source. 'L' The federated server folds the password to lowercase before sending it to the data source. If none of these settings are used, the federated server tries to send the password to the data source in uppercase. If the password fails, the server tries sending it in lowercase.	None.
io_ratio	Denotes how much faster or slower a data source's I/O system runs than the federated server's I/O system.	'1.0'

Table 22. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
node	Name by which a data source is defined as an instance to its RDBMS. Required for all data sources. For a DB2 family data source, this name is the node specified in the federated database's DB2 node directory. To view this directory, issue the **db2 list node directory** command. For an Oracle** data source, this name is the server name specified in the Oracle** tnsnames.ora file. To access this name on the Windows NT platform, specify the **View Configuration Information** option of the Oracle** SQL Net Easy Configuration tool.	None.
password	Specifies whether passwords are sent to a data source. 'Y' Passwords are always sent to the data source and validated. This is the default value. 'N' Passwords are not sent to the data source (regardless of any user mappings) and not validated. 'ENCRYPTION' Passwords are are always sent to the data source in encrypted form and validated. Valid only for DB2 Family data sources that support encrypted passwords.	'Y'
plan_hints	Specifies whether *plan hints* are to be enabled. Plan hints are statement fragments that provide extra information for data source optimizers. This information can, for certain query types, improve query performance. The plan hints can help the data source optimizer decide whether to use an index, which index to use, or which table join sequence to use. 'Y' Plan hints are to be enabled at the data source if the data source supports plan hints. 'N' Plan hints are not to be enabled at the data source.	'N'
pushdown	'Y' DB2 will consider letting the data source evaluate operations. 'N' DB2 will retrieve only columns from the remote data source and will not let the data source evaluate other operations, such as joins.	'Y'

Table 22. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
varchar_no_trailing_blanks	Specifies if this data source uses non-blank padded varchar comparison semantics. For varying-length character strings that contain no trailing blanks, some DBMS' s non-blank-padded comparison semantics return the same results as DB2's comparison semantics. If you are certain that all VARCHAR table/view columns at a data source contain no trailing blanks, consider setting this server option to 'Y' for a data source. This option is often used with Oracle** data sources. Ensure that you consider all objects that can potentially have nicknames (including views). 'Y' This data source has non-blank-padded comparison semantics similar to DB2's. 'N' This data source does not have the same non-blank-padded comparison semantics as DB2's.	'N'

Notes on this table:

1. This field is applied regardless of the value specified for authentication.
2. Because DB2 stores user IDs in uppercase, the values 'N' and 'U' are logically equivalent to each other.
3. The setting for fold_pw has no effect when the setting for password is 'N'. Because no password is sent, case cannot be a factor.
4. Avoid null settings for either of these options. A null setting may seem attractive because DB2 will make multiple attempts to resolve user IDs and passwords; however, performance might suffer (it is possible that DB2 will send a user ID and password four times before successfully passing data source authentication).

Using Pass-through Sessions with Servers: Pass-through sessions let applications communicate directly with a server using the server's native client access method and native SQL dialect.

Pass-through sessions are useful when:

- Applications must create objects at the data source or perform INSERT, UPDATE, or DELETE operations
- DB2 does not support a unique data source operation

When referencing objects in a pass-through session, use the true name of the object (not the nickname).

Use the SET PASSTHRU statement to start a pass-through session and access a server directly. This statement must be issued dynamically. An example of this statement is:

```
SET PASSTHRU BACKEND
```

which opens a pass-through session to the data source BACKEND.

For more information on SET PASSTHRU and SQL processing in pass-through sessions, see the *SQL Reference*.

Creating a Nickname

In a federated database, nicknames are identifiers for data source tables, aliases, and views. Distributed requests typically reference nicknames, not data source tables or views.

Nicknames are part of the means by which DB2 provides location transparency. Nicknames rely on server definitions for data source location information to find and efficiently access data sources. An ALTER SERVER statement can, for example, transparently update server performance data and version information for all users and applications without requiring new nicknames or changes to application code.

Nicknames can be created in the Control Center or from the command line processor. You can define more than one nickname for the same data source table or view.

Nicknames cannot be used in static SQL statements.

Before creating a nickname, run the equivalent of the RUNSTATS command at the data source and update statistics for data source objects. Statistical information is gathered from data sources when a nickname is created and stored in the federated database catalog. This catalog data includes table and column definitions, and, if available, index definitions and statistics.

The following SQL statement creates the nickname CUSTOMER:

```
CREATE NICKNAME CUSTOMER for OS390A.SHAWNB.CUSTLIST
```

You must hold one of the SYSADM or DBADM authorities, or, you must have either the database privilege IMPLICIT_SCHEMA or the schema privilege CREATEIN (for the current schema) at the federated database to use this statement.

For additional details on using the CREATE NICKNAME statement, refer to the *SQL Reference*.

Referencing Nickname and Data Source Objects

References to data source objects typically use the defined nickname. The one exception is a reference within a pass-through session (see “Using Pass-through Sessions with Servers” on page 197 for more information). For example, if you define the nickname DEPT for the data source table DB2MVS1.PERSON.DEPT, the statement SELECT * FROM DEPT is allowed; the statement SELECT * FROM DB2MVS1.PERSON.DEPT is not allowed.

Working with Nickname and Data Source Objects

Most utility commands (LOAD, IMPORT, EXPORT, REORGCHK, REORGANIZE TABLE) do not support nicknames

COMMENT ON is supported; it updates the system catalog at the federated database.

INSERT, UPDATE, and DELETE operations are not supported against nicknames.

Identifying Existing Nicknames and Data Sources

After you have created several nicknames, you might want to use the following information to identify to which data source a given nickname corresponds or identify all nicknames at a given data source.

Identifying a Nickname and Its Data Source: This example assumes that you know the nickname (*PAYROLL*) and who created it (*ACCTG*), but need additional information about the data source. Use the following SQL statement to first obtain information about what *PAYROLL* is known as at its data source (SERVER).

```
select option, setting
  from syscat.taboptions
  where tabname = 'PAYROLL'
     and tabschema = 'ACCTG'
     and option in ('SERVER','REMOTE_SCHEMA','REMOTE_TABLE');
```

The answer set from this statement is `DB2_MVS, FINANCE, DEPTJ35_PAYROLL`. You now know that *PAYROLL* is the nickname for the table called DEPTJ35_PAYROLL owned by FINANCE at the server named DB2_MVS. You can use this information in a subsequent SELECT statement:

```
 select option,setting
     from syscat.serveroptions
     where servername = 'DB2_MVS'
        and option in ('NODE','DBNAME');
```

The answer set from this statement is REGIONW and DB2MVSDB3. You now know that the table DEPTJ35_PAYROLL is in a database named DB2MVSDB3, on a node called REGIONW.

With this information, you can use the LIST NODE DIRECTORY command to obtain information about the REGIONW node, such as the communications protocol and security type used. If the node had been for a data source other than the DB2 Family, you would need to check that data source's configuration files to find similar information. For example, if the node had been an Oracle data source, you would get similar information from the Oracle tnsnames.ora file.

For details on system catalog views, refer to the *SQL Reference*.

Identifying All Nicknames Known to DB2: The following SQL statement provides a list of all nicknames known to the federated database, including the schema name and remote server for each nickname.

```
select tabname,tabschema, setting as remote_server
   from syscat.taboptions
   where option = 'SERVER';
```

Creating an Index or an Index Specification

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 specification is a metadata construct. It tells the optimizer that an index exists for a data source object (table or view) referenced by a nickname. An index specification does not contain lists of row locations–it is just a description of an index. The optimizer uses the index specification to improve access to the object referenced by the nickname. When a nickname is first created, an index specification is generated if an index exists for the underlying table at the data source in a format DB2 can recognize.

Note: If needed, create index specifications on table nicknames or view nicknames where the view is over one table.

Manually create an index or an index specification when:

- It would improve performance. For example, if you want to encourage the optimizer to use a particular table or nickname as the inner table of a nested loop join, create an index specification on the joining column if no

index exists. See the *Administration Guide, Performance* for more information about when you would want an index or an index specification.

- An index for a base table was added after the nickname for that table was created.

Index specifications can be created when no index exists on the base table (DB2 will not check for the remote index when you issue the CREATE INDEX statement). An index specification does not enforce uniqueness of rows even when the UNIQUE keyword is specified.

The DB2 Index Advisor is a wizard that assists you in choosing an optimal set of indexes. You can access this wizard through the Control Center. The comparable utility is called *db2advis*.

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. 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.

For a *clustering index*, new rows are inserted physically close to existing rows with similar key values. This yields a performance benefit during queries because it results in a more linear access pattern to data pages and more effective pre-fetching.

If you want a primary key index to be a clustering index, a primary key should not be specified at CREATE TABLE. Once a primary key is created, the associated index cannot be modified. Instead, perform a CREATE TABLE without a primary key clause. Then issue a CREATE INDEX statement,

specifying clustering attributes. Finally, use the ALTER TABLE statement to add a primary key that corresponds to the index just created. This index will be used as the primary key index.

Generally, clustering is more effectively maintained if the clustering index is unique.

Column data which is not part of the unique index key but which is to be stored / maintained in the index is called an *include* column. Include columns can be specified for unique indexes only. When creating an index with include columns, only the unique key columns are sorted and considered for uniqueness. Use of include columns improves the performance of data retrieval when index access is involved.

The database manager uses a B+ tree structure for storing indexes where the bottom level consists of `leaf nodes`. The leaf nodes or pages are where the actual index key values are stored. When creating an index, you can enable those index leaf pages to be merged or reorganized online. online index reorganization is used to prevent the situation where, after much delete and update activity, many leaf pages of an index have only a few index keys left on them. In such a situation, and without online reorganization, space could only be reclaimed by an off-line reorganization of the data and index. When deciding whether to create an index with the ability to reorganize index pages online, you should consider the following:

- Is the added performance cost of checking for space to merge each time a key deletion occurs; and the actual cost to complete the merge, if there is enough space
- Greater than the benefit of better space utilization for the index; and less need to perform an off-line reorganization to reclaim space.

Note: Pages freed after an online reorganization merge are available for re-use only for other indexes in the same table. With a full reorganization, those pages that are freed are available to other objects (when working with Database Managed Storage) or to disk space (when working with System Managed Storage). In addition, an online reorganization will not free up any non-leaf pages of the index, whereas a full reorganization will make the index as small as possible by making the index as small as possible, reducing the non-leaf and leaf pages as well as the number of levels of the index.

See “Using the CREATE INDEX Statement” on page 204 for more information on how to implement an index that will reorganize online.

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 consists 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 `statistics yes` option requested.

For more information on LOAD performance improvements, see "System Catalog Tables" on page 59.

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, unless there is a clustering index defined, the row 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 172).

The following two sections, "Using an Index" and "Using the CREATE INDEX Statement" on page 204, 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, refer to "Index Scan Concepts" in the *Administration Guide, Performance.*

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.

When working with a structured type, it might be necessary to create user-defined index types. This requires a means of defining index maintenance, index search, and index exploitation functions. Refer to the *SQL Reference* for information on the requirements for creating an index type.

The following SQL statement creates a clustering index called INDEX1 on LASTNAME column of the EMPLOYEE table:

```
CREATE INDEX INDEX1 ON EMPLOYEE (LASTNAME) CLUSTER
```

To be effective, use clustering indexes with the PCTFREE parameter associated with the ALTER TABLE statement so that new data can be inserted on the correct pages which maintains the clustering order. Typically, the greater the INSERT activity on the table, the larger the PCTFREE value (on the table) that will be needed in order to maintain clustering. Since this index determines the

order by which the data is laid out on physical pages, only one clustering index can be defined for any particular table.

If, on the other hand, the index key values of these new rows are, for example, always new high key values, then the clustering attribute of the table will try to place them at the end of the table. Having free space in other pages will do little to preserve clustering. In this case, placing the table in append mode may be a better choice than a clustering index and altering the table to have a large PCTFREE value. You can place the table in append mode by issuing: ALTER TABLE APPEND ON. See “Changing Table Attributes” on page 222 for additional overview information on ALTER TABLE. Refer to the *SQL Reference* for additional detailed information on ALTER TABLE.

The above discussion also applies to new ″overflow″ rows that result from UPDATEs which increase the size of a row.

The MINPCTUSED clause of the CREATE INDEX statement specifies the threshold for the minimum amount of used space on an index leaf page. If this clause is used, online index reorganization is enabled for this index. Once enabled, the following considerations are used to determine if an online reorganization takes place: After a key is deleted from a leaf page of this index and a percentage of used space on the page is less than the specified threshold value, the neighboring index leaf pages are checked to determine if the keys on the two leaf pages can be merged into a single index leaf page.

For example, the following SQL statement creates an index with online index reorganization enabled:

```
CREATE INDEX LASTN ON EMPLOYEE (LASTNAME) MINPCTUSED=20
```

When a key is deleted from this index, if the remaining keys on the index page take up twenty percent or less space on the index page, then an attempt is made to delete an index page by merging the keys of this index page with those of a neighboring index page. If the combined keys can all fit on a single page, this merge is performed and one of the index pages is deleted.

The PCTFREE clause of the CREATE INDEX statement specifies the percentage of each index page to leave as free space when the index is built. Leaving more free space on the index pages will result in fewer page splits. This will reduce the need to reorganize the table in order to regain sequential index pages which increases prefetching. And prefetching is one important component that may improve performance. Again, if there are always high key values, then you will want to consider lowering the value of the PCTFREE clause of the CREATE INDEX statement. In this way there will be limited wasted space reserved on each index page.

In multiple partition databases, unique indexes must be defined as supersets of the partitioning key.

If you have a replicated summary table, its base table (or tables) must have a unique index, and the index key columns must be used in the query that defines the replicated summary table. For more information, see "Replicated Summary Tables" on page 74.

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

Some time after a database design has been implemented, you may be considering a change to the database design. As a result, you should reconsider the major design issues that you had with the previous design. You should consider the following:

- Changing Logical and Physical Design Characteristics
- Changing the License Information
- Changing Instances
- Changing Environment Variables and the Profile Registry Variables
- Changing the Node Configuration File
- Changing the Database Configuration

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 75.)

Changing the License Information

As part of the management of licenses for your DB2 products, you may find that you have a need to increase the number of licenses. You can use the

License Center within the Control Center to check usage of the installed products and increase the number of licenses based on that usage.

Changing Instances

Existing instances are designed to be as independent as possible from the effects of subsequent installation and removal of products.

In most cases, existing instances automatically inherit or lose access to the function of the product being installed or removed. However, if certain executables or components are installed or removed, existing instances do not automatically inherit the new system configuration parameters or gain access to all the additional function. The instance must be updated.

If DB2 is updated by installing a Program Temporary Fix (PTF) or a patch, all the existing DB2 instances should be updated using the **db2iupdt** command. You should also update the Administration Server (DAS) using the **dasiupdt** command.

You should ensure you understand the instances and database partition servers you have in an instance before attempting to change or delete an instance.

Listing Instances

To get a list of all the instances that are available on a system, enter:

```
db2ilist
```

To determine which instance applies to the current session,

```
set db2instance
```

Listing Database Partition Servers in an Instance

Use the **db2nlist** command to obtain a list of database partition servers that participate in an instance.

```
db2nlist
```

When using this command as shown, the default instance is the current instance (set by the DB2INSTANCE environment variable). To specify a particular instance, you can specify the instance using:

```
db2nlist /i:instName
```

where `instName` is the particular instance name you want.

You can also optionally request the status of each partition server by using:

```
db2nlist /s
```

The status of each database partition server may be one of: starting, running, stopping, or stopped.

Updating Instances

Running the **db2iupdt** command updates the specified instance by performing the following:

- Replaces the files in the `sqllib` subdirectory under the instance owner's home directory.
- If the node type is changed, then a new database manager configuration file is created. This is done by merging relevant values from the existing database manager configuration file with the default database manager configuration file for the new node type. If a new database manager configuration file is created, the old file is backed up to the `backup` subdirectory of the `sqllib` subdirectory under the instance owner's home directory.

The **db2iupdt** command is found in the `instance` subdirectory in the version and release subdirectory (the exact name varies by operating system).

The command is used as shown:

```
db2iupdt InstName
```

The `InstName` is the log in name of the instance owner.

There are other optional parameters associated with this command:

- –h or –?

  Displays a help menu for this command.
- –d

  This parameter sets the debug mode for use during problem determination.
- –a AuthType

  This parameter specifies the authentication type for the instance. Valid authentication types are SERVER, CLIENT, DCS, or DCE. If not specified, the default is SERVER, if a DB2 server is installed. Otherwise, it is set to CLIENT.

  Notes:

  1. The authentication type of the instance applies to all databases owned by the instance.
  2. On UNIX operating systems, the authentication type DCE is not a valid choice.
- –u FencedID

This parameter is the user under which the fenced user-defined functions (UDFs) and stored procedures will execute. This is not required if you install the DB2 client or the DB2 Software's Developer Kit. For other DB2 products, this is a required parameter.

Note: FencedID may not be "root" or "bin".

- –k

  This parameter preserves the current instance type. If you do not specify this parameter, the current instance is upgraded to the highest instance type available in the following order:

  – Partitioned database server with local and remote clients (DB2 Extended Enterprise Edition default instance type)
  – Database Server with local and remote clients (DB2 Universal Database Enterprise Edition default instance type)
  – Client (DB2 client default instance type)

Examples:

- If you installed DB2 Universal Database Workgroup Edition or DB2 Universal Database Enterprise Edition after the instance was created, enter the following command to update that instance:

  ```
  db2iupdt -u db2fenc1 db2inst1
  ```

- If you installed the DB2 Connect Enterprise Edition after creating the instance, you can use the instance name as the Fenced ID also:

  ```
  db2iupdt -u db2inst1 db2inst1
  ```

- To update client instances, you can use the following command:

  ```
  db2iupdt db2inst1
  ```

Adding a Database Partition Server to an Instance

Use the **db2ncrt** command to add a database partition server (node) to an instance.

Note: Do not use the **db2ncrt** command if there are databases already existing in this instance. Instead, use the **db2start addnode** command. This ensures that the database is correctly added to the new database partition server. **DO NOT EDIT** the `db2nodes.cfg` file, since changing the file may cause inconsistencies in the partitioned database system.

The command has the following required parameters:

```
db2ncrt /n:node_number
        /u:username,password
```

The first parameter (/n:) is the unique node number to identify the database partition server. The number can be from 1 to 999 in ascending sequence.

The second parameter (/u:) is the logon account name and password of the DB2 service.

There are also several optional parameters:

- /l:instance_name

  The instance name; the default is the current instance.
- /m:machine_name

  The computer name of the Windows NT workstation on which the node resides; the default name is the computer name of the local machine.
- /p:logical_port

  Specifies the logical port number used for the database partition server if the logical port is not zero (0).
- /h:host_name

  The TCP/IP host name that is used by FCM for internal communications if the host name is not the local host name.
- /o:instance_owning_machine

  The computer name of the machine that is the instance-owning machine; the default is the local machine. This parameter is required when the **db2ncrt** command is invoked on any machine that is not the instance-owning machine.

Dropping a Database Partition Server from an Instance

Use the **db2ndrop** command to drop a database partition server (node) from an instance that has no databases. If you drop a database partition server, its node number can be reused for a new database partition server.

Exercise caution when you are dropping database partition servers from an instance. If you drop the instance-owning database partition server node zero (0) from the instance, the instance will become unusable. If you want to drop the instance, use the **db2idrop** command.

Note: Do not use the **db2ndrop** command if there are databases in this instance. Instead, use the **db2stop drop** command. This ensures that the database is correctly removed from the database partition server. **DO NOT EDIT** the db2nodes.cfg file, since changing the file may cause inconsistencies in the partitioned database system.

The command has the following required parameter:

```
db2ndrop /n:node_number
```

The parameter (/n:) is the unique node number to identify the database partition server. The number can be from zero (0) to 999 in ascending sequence. Recall that node zero (0) represents the instance-owning machine.

There is an optional parameter (/i:instance_name) which is the instance name; the default is the current instance (set by the DB2INSTANCE environment variable).

Removing Instances

To remove an instance, perform the following steps:

1. End all applications that are currently using the instance.
2. Stop the Command Line Processor by running **db2 terminate** commands in each DB2 command window.
3. Stop the instance by running the **db2stop** command.
4. Back up the instance directory indicated by the DB2INSTPROF registry variable. On UNIX operating systems, you might consider backing up the files in the `INSTHOME/sqllib` directory (where `INSTHOME` is the home directory of the instance owner). For example, you might want to save the database manager configuration file, `db2systm`, the `db2nodes.cfg` file, user-defined functions (UDFs), or fenced stored procedure applications.
5. (On UNIX operating systems only) Log off as the instance owner.
6. (On UNIX operating systems only) Log in as a user with root authority.
7. Issue the **db2idrop** command:

   ```
   db2idrop InstName
   ```

   where `InstName` is the name of the instance being dropped.

   This command removes the instance entry from the list of instances and removes the instance directory.
8. (On UNIX operating systems only) Optionally, as a user with root authority, remove the instance owner's user ID and group (if used only for that instance). Do not remove these if you are planning to re-create the instance.

   This step is optional since the instance owner and the instance owner group may be used for other purposes.

The **db2idrop** command removes the instance entry from the list of instances and removes the `sqllib` subdirectory under the instance owner's home directory.

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 and you are not on a UNIX platform, 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 variables to those that are best suited to the new database environment. If only profile registry variables have been changed, the system does not need to be restarted.

Changing the Node Configuration File

If you are planning changes to any nodegroups (both adding or deleting nodes, or moving existing nodes), you should refer to "Scaling Your Configuration" in the *Administration Guide, Performance* for details on what should be done.

Changing the Database Configuration

If you are planning changes to the database, you should review the values for the configuration parameters. Some of the values can be adjusted from time-to-time as part of the ongoing changes made to the database based on how it is used.

To change the database configuration, use the Performance Configuration SmartGuide. This SmartGuide 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.
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, refer to "Benchmark Testing" and "Configuring DB2" in the *Administration Guide, Performance*.

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.

Altering a Database

There are nearly as many tasks when altering databases as there are in the creation of databases. These tasks update or drop aspects of the database previously created. The tasks include:

- Dropping a Database
- Altering a Nodegroup
- Altering a Table Space
- Dropping a Schema
- Modifying a Table in Both Structure and Content
- Altering a User-Defined Structured Type
- Deleting and Updating Rows of a Typed Table
- Renaming an Existing Table
- Dropping a Table
- Dropping a Trigger
- Dropping a User-Defined Function (UDF), Function Template, or Function Mapping
- Dropping a User-Defined Type (UDT) or Type Mapping
- Altering or Dropping a View
- Dropping a Summary Table
- Altering or Dropping a Server
- Altering or Dropping a Nickname

- Dropping an Index or an Index Specification
- Statement Dependencies When Changing Objects.

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
```

Note: If you intend to continue experimenting with the SAMPLE database, you should not drop it. If you have dropped the SAMPLE database, and find that you need it again, you can re-create it.

Altering a Nodegroup

To add or drop database partitions from a nodegroup, you can use the ALTER NODEGROUP statement. When adding database partitions, the partitions must already be defined in the node configuration file. Refer to the *SQL Reference* for details on this statement.

To add a new node to the `db2nodes.cfg` file, use the START DATABASE MANAGER command or `dbstart`. The `db2nodes.cfg` file is not updated with the new node until a `db2stop` followed by a `db2start`. Refer to the *Command Reference* for details on this statement.

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, refer to “Redistributing Data Across Database Partitions” in the *Administration Guide, Performance* and to the *Command Reference.*

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: You cannot drop the catalog table space SYSCATSPACE, and there must always be at least one temporary table space. You also cannot change the page size.

This section discusses how to change table spaces as follows:

- "Adding a Container to a DMS Table Space"
- "Dropping a User Table Space"
- "Dropping a Temporary Table Space" on page 216.

See "Designing and Choosing Table Spaces" on page 75 for design information on table spaces.

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. Refer to "Table Space Impact on Query Optimization" in the *Administration Guide, Performance* for more information.

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 can drop a user table space that contains all of the table data including index and LOB data within that single user table space. You can also drop a user table space that may have tables spanned across several table spaces. That is, you may have table data in one table space, indexes in another, and any LOBs in a third table space. You may drop each table space individually as long as the table space with the table data is dropped first. Or, you may drop all three table spaces at the same time in a single statement. All of the table spaces that contain tables that are spanned must be part of this single statement or the drop request will fail. Refer to the *SQL Reference* for details on how to drop table spaces containing spanned table data.

The following SQL statement drops the table space ACCOUNTING:

```
DROP TABLESPACE ACCOUNTING
```

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 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, for example, add a container to an SMS Temporary 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. 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
```

Modifying a Table in Both Structure and Content

Tasks that are required for modifying the structure and content of the table include the following:

- Adding Columns to an Existing Table
- Modifying a Column
- Altering a Constraint
- Adding a Constraint
- Dropping a Constraint
- Declaring a Table Volatile

- Changing Partitioning Keys
- Changing Table Attributes
- Refreshing the Data in a Summary Table.

Note that you cannot alter triggers for tables; you must drop any trigger that is no longer appropriate (see “Dropping a Trigger” on page 225), and add its replacement (see “Creating a Trigger” on page 174).

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.

Modifying a Column

You can modify the characteristics of a column by increasing the length of an existing VARCHAR column. The number of characters may increase up to a value dependent on the page size used. For example, to increase a column up to 4000 characters, use something similar to the following:

```
ALTER TABLE ALTER COLUMN
   COLNAM1 SET DATA TYPE VARCHAR(4000)
```

You cannot alter the column of a typed table. However, you can add a scope to an existing reference type column that does not already have a scope defined. For example:

```
ALTER TABLE ALTER COLUMN
   COLNAMT1 ADD SCOPE TYPTAB1
```

For more information about the ALTER TABLE statement, refer to 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 219

For more information on constraints, see "Defining Constraints" on page 162.

Adding a Constraint

You add constraints with the ALTER TABLE statement. For more information on this statement, including its syntax, refer to the *SQL Reference* manual.

For more information on constraints, see "Defining Constraints" on page 162.

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 INTEGRITY 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 233 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 INTEGRITY 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 233 for more information.

Dropping a Constraint

You drop constraints with the ALTER TABLE statement. For more information on this statement, including its syntax, refer to the *SQL Reference* manual.

For more information on constraints, see "Defining Constraints" on page 162.

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, refer to 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 233 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 233 for more information.) To drop a table check constraint with a system-generated name, look for the name in the SYSCAT.CHECKS catalog view.

Declaring a Table Volatile

A *volatile* table is defined as a table whose contents can vary from empty to very large at run time. The volatile or extreme changeability of this type of table makes reliance on the statistics collected by RUNSTATS to be inaccurate. Statistics are gathered at, and only reflect, a point-in-time. To generate an access plan that uses a volatile table can result in an incorrect or poor performing plan. For example, if statistics are gathered when the volatile table is empty, the optimizer tends to favor accessing the volatile table using a table scan rather than an index scan.

To prevent this, you should consider declaring the table as "volatile" using the ALTER TABLE statement. By declaring the table "volatile", the optimizer will consider using index scan rather than table scan. The access plans that use declared volatile tables will not depend on the existing statistics for that table.

The way to declare a table as "volatile" is to do the following:

```
ALTER TABLE TABLENAME
   VOLATILE CARDINALITY
```

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
```

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.

Changing Table Attributes

You may have reason to change table attributes such as the data capture option, the percentage of free space on each page (PCTFREE), the lock size, or the append mode.

The amount of free space to be left on each page of a table is specified through PCTFREE, and is an important consideration for the effective use of clustering indexes. The amount to specify depends on the nature of the existing data and expected future data. PCTFREE is respected by LOAD and REORG but is ignored by insert, update and import activities.

Setting PCTFREE to a larger value will maintain clustering for a longer period, but will also require more disk space.

You can specify the size (granularity) of locks used when the table is accessed by using the LOCKSIZE parameter. By default, when the table is created row level locks are defined. Use of table level locks may improve the performance of queries by limiting the number of locks that need to be acquired and released.

By specifying APPEND ON, you can improve the overall performance. It allows for faster insertions, while eliminating the maintenance of information about the free space.

A table with a clustering index cannot be altered to have append mode turned on. Similarly, a clustering index cannot be created on a table with append mode.

Refreshing the Data in a Summary Table

You can refresh the data in a summary table by using the REFRESH TABLE statement. The statement can be embedded in an application program, or issued dynamically. To use this statement, you must have either SYSADM or DBADM authority, or CONTROL privilege on the table to be refreshed.

The following example shows how to refresh the data in a summary table:

```
REFRESH TABLE SUMTAB1
```

For more information about the REFRESH TABLE statement, refer to the *SQL Reference*.

Altering a User-Defined Structured Type

After creating a structured type, you may find that you need to add or drop attributes associated with that structured type. This is done using the ALTER TYPE (Structured) statement.

Note: You are not allowed to modify a type if any table exists of that type.

For example, you may find you need to add an attribute to an existing row type:

```
ALTER TYPE Employee_t ADD ATTRIBUTE DeptNum INT;
```

This example adds a new attribute, `DeptNum`, to the `Employee_t` structured type. Note that ALTER TYPE is only permitted on structured types that are not currently in use as the type of an existing table or subtable.

In a similar way, you might also consider altering a typed table or view. However, the only change that is permitted to a typed table or view is to specify the scope of a reference-type column that does not yet have a scope.

Refer to the *SQL Reference* for more information on the ALTER TYPE(Structured), ALTER TABLE, and ALTER VIEW statements.

Deleting and Updating Rows of a Typed Table

Rows can be deleted from typed tables using either searched or positioned DELETE statements. In addition, since a typed table may have subtables, the ONLY option may be used in the FROM clause if it is desirable to avoid having subtable rows affected by the delete operation. This is applicable to both typed tables and typed views.

Refer to the *SQL Reference* for more information on the DELETE statement.

Rows can be updated in typed tables using either searched or positioned UPDATE statements. In addition, since a typed table may have subtables, the ONLY option may be used in the FROM clause if it is desirable to avoid having subtable rows affected by the update operation. This is applicable to both typed tables and typed views.

Refer to the *SQL Reference* for more information on the UPDATE statement.

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 to be renamed can be an alias identifying a table. The existing table to be renamed must not be the name of a catalog table, a summary table, a typed table, or an object other than a table or an alias.

The existing table cannot be referenced in any of the following:

- Views
- Triggers
- Referential constraints
- Summary table
- The scope of an existing reference column.

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, refer to 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 DEPARTMENT:

```
DROP TABLE DEPARTMENT
```

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 228 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. This includes packages dependent on any supertable above the subtable in the hierarchy that is being dropped. (See “Statement Dependencies When Changing Objects” on page 233 for more information.)
- Any reference columns for which the dropped table is defined as the scope of the reference become “unscoped”.
- 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.
- All files that are linked through any DATALINK columns are unlinked. The unlink operation is performed asynchronously which means the files may not be immediately available for other operations.

An individual table cannot be dropped if it has a subtable. However, all the tables in a table hierarchy can be dropped by a single DROP TABLE HIERARCHY statement, as in the following example:

```
DROP TABLE HIERARCHY person
```

The DROP TABLE HIERARCHY statement must name the root table of the hierarchy to be dropped.

There are differences when dropping a table hierarchy compared to dropping a specific table:

- DROP TABLE HIERARCHY does not activate deletion-triggers that would be activated by individual DROP table statements. For example, dropping an individual subtable would activate deletion-triggers on its supertables.
- DROP TABLE HIERARCHY does not make log entries for the individual rows of the dropped tables. Instead, the dropping of the hierarchy is logged as a single event.

Refer to the *SQL Reference* for more information on the DROP statement.

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), Function Template, or Function Mapping

A user-defined function (UDF), function template, or function mapping can be dropped using the DROP statement.

You can disable a function mapping with the mapping option DISABLE. Refer to the *SQL Reference* for more information on how to do this.

A UDF cannot be dropped if a view, trigger, table check constraint, or another UDF is dependent on it. 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 or function template. All such dependencies, including function mappings, 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. Refer to 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 function mapping marks a package as invalid. Automatic rebind will take place and the optimizer will attempt to use the local function. In the case where the local function is a template, the implicit rebind will fail.

(For more information, see “Statement Dependencies When Changing Objects” on page 233.)

Dropping a User-Defined Type (UDT) or Type Mapping

You can drop a user-defined type (UDT) or type mapping using the DROP statement. You cannot drop a UDT if it is used:

- In a column definition for an existing table or view (distinct types)
- As the type of an existing typed table or typed view (structured type)
- As the supertype of another structured type.

You cannot drop a default type mapping; you can only override it by creating another type mapping.

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.

If you have created a transform for a UDT, and you are planning to drop the UDT, you should consider if it is necessary to drop the transform. This is done through the DROP TRANSFORM statement. Refer to the *SQL Reference* for details on this statement. Note that only transforms defined by you or other application developers can be dropped; built-in transforms and their associated group definitions cannot be dropped.

For more information about the user-defined types, refer to the *SQL Reference* and *Application Development Guide* manuals.

Altering or Dropping a View

The ALTER VIEW statement modifies an existing view by altering a reference type column to add a scope. Any other changes you make to a view require that you drop and then re-create the view.

When altering the view, the scope must be added to an existing reference type column that does not already have a scope defined. Further, the column must not be inherited from a superview.

The data type of the column-name in the ALTER VIEW statement must be REF (type of the typed table name or typed view name).

Refer to the *SQL Reference* for additional information on the ALTER VIEW statement.

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 233 for more information.

As in the case of a table hierarchy, it is possible to drop an entire view hierarchy in one statement by naming the root view of the hierarchy, as in the following example:

```
DROP VIEW HIERARCHY VPerson
```

For more information on dropping and creating views, refer to the *SQL Reference* manual.

Recovering Inoperative Views

Views can become *inoperative* as a result of a revoked SELECT privilege on an underlying table.

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 a Summary Table

You cannot alter a summary table, but you can drop it. The following SQL statement drops the summary table XT:

```
DROP TABLE XT
```

All indexes, primary keys, foreign keys, and check constraints referencing the table are dropped. All views and triggers that reference the table are made inoperative. All packages depending on any object dropped or marked inoperative will be invalidated. See "Statement Dependencies When Changing Objects" on page 233 for more information on package dependencies.

Recovering Inoperative Summary Tables

Summary tables can become *inoperative* as a result of a revoked SELECT privilege on an underlying table.

The following steps can help you recover an inoperative summary table:

- Determine the SQL statement that was initially used to create the summary table. You can obtain this information from the TEXT column of the SYSCAT.VIEW catalog view.
- Re-create the summary table by using the CREATE SUMMARY TABLE statement with the same summary table name and same definition.
- Use the GRANT statement to re-grant all privileges that were previously granted on the summary table. (Note that all privileges granted on the inoperative summary table are revoked.)

If you do not want to recover an inoperative summary table, you can explicitly drop it with the DROP TABLE statement, or you can create a new summary table with the same name but a different definition.

An inoperative summary table 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 a Wrapper

The DROP statement can remove a wrapper from the database. The following example shows how to drop the DRDA wrapper:

```
DROP WRAPPER DRDA
```

Any servers, type mappings, function mappings, user mappings, and nicknames that are dependent on the wrapper being dropped will be dropped. Exercise extreme caution when dropping wrappers.

You must hold one of the SYSADM or DBADM authorities to DROP wrappers.

Refer to the *SQL Reference* for more information on dropping wrappers.

Altering or Dropping a Server

The ALTER SERVER statement modifies an existing server definition in the federated database catalog. Use this statement to:

- Modify the definition of a specific data source.
- Modify the definition of multiple data sources of a specific type or version.
- Make changes in the configuration of a specific data source. For example, if the DBMS identified by a specific server is migrated to a new workstation with a faster processor, you should update the cpu_ratio server option.

You cannot use this statement to modify the *dbname* or *node* server options.

The following example shows how to alter the ORA1 server:

```
ALTER SERVER ORA1 OPTIONS (SET CPU_RATIO '5.0')
```

Servers can be dropped from the federated database. The following example shows how to drop the ORALOC01 Server:

```
DROP SERVER ORALOC01
```

Any type mappings, function mappings, user mappings, and nicknames that are dependent on the server being dropped will be dropped. Exercise caution when dropping servers.

You must hold one of the SYSADM or DBADM authorities to ALTER or DROP servers.

For more information on dropping and altering servers, refer to the *SQL Reference*.

Altering or Dropping a Nickname

The ALTER NICKNAME statement is used to update locally stored information about a data source table or view. You could use this statement, for example, to change the local name for a column or to map a column data type to a different data type. You can also use this statement to add column options. For more information on ALTER NICKNAME syntax, see the *SQL Reference*.

When a nickname is dropped, views created on that nickname are marked as inoperative. You cannot alter nickname column names or data types when the nickname is referenced in a view.

You must hold one of the SYSADM or DBADM authorities, or, you must have either the CONTROL or ALL database privilege on the nickname, the

ALTERIN (for the current schema) schema privilege, or be the nickname definer at the federated database to use this statement.

Altering a Nickname Column and Dropping a Nickname

The following example shows how to alter the nickname TESTNN, changing the local name of a column from COL1 to NEWCOL:

```
ALTER NICKNAME TESTNN ALTER COLUMN COL1 LOCAL NAME NEWCOL
```

The following example shows how to drop the nickname TESTNN:

```
DROP NICKNAME TESTNN
```

Altering Nickname Column Options

You specify column information in the form of values that you assign to parameters called *column options*. You can specify any of these values in either upper- or lowercase. The table below describes the values and provides additional information.

Table 23. Column Options and Their Settings

Option	Valid Settings		Default Setting
numeric_string	'Y'	Yes, this column contains only strings of numeric data. IMPORTANT: If this column contains only numeric strings followed by trailing blanks, it is inadvisable to specify 'Y'.	'N'
	'N'	No, this column is not limited to strings of numeric data.	
	By setting numeric_string to 'Y' for a column, you are informing the optimizer that this column contains no blanks that could interfere with sorting of the column's data. This option is helpful when the collating sequence of a data source is different from DB2. Columns marked with this option will not be excluded from local (data source) evaluation because of a different collating sequence.		

Table 23. Column Options and Their Settings (continued)

Option	Valid Settings	Default Setting
varchar_no_trailing_blanks	Indicates whether trailing blanks are absent from a specific VARCHAR column: 'Y' Yes, trailing blanks are absent from this VARCHAR column. 'N' No, trailing blanks are not absent from this VARCHAR column. If data source VARCHAR columns contain no padded blanks, then the optimizer's strategy for accessing them depends in part on whether they contain trailing blanks. By default, the optimizer "assumes" that they actually do contain trailing blanks. On this assumption, it develops an access strategy that involves modifying queries so that the values returned from these columns are the ones that the user expects. If, however, a VARCHAR column has no trailing blanks, and you let the optimizer know this, it can develop a more efficient access strategy. To tell the optimizer that a specific column has no trailing blanks, specify that column in the ALTER NICKNAME statement (for syntax, see the *SQL Reference*).	'N'

Dropping an Index or an Index Specification

You cannot change any clause of an index definition or specification; you must drop the index and create it again. (Dropping an index or an index specification 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 (unless it is an index specification) 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.

Any packages and cached dynamic SQL statements that depend on the dropped 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 *Application Development Guide* discusses how to create packages in detail.

Packages and cached dynamic SQL statements can be dependent on many types of objects. Refer to the *SQL Reference* for a complete list of those objects.

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 is placed in an ″invalid″ state. If a package depends on a user-defined function and that function is dropped, the package is placed in an ″inoperative″ state.

A cached dynamic SQL statement that is in an invalid state is automatically re-optimized on its next use. If an object required by the statement has been dropped, execution of the dynamic SQL statement may fail with an error message.

A package that is in an invalid state is implicitly rebound on its next use. Such a package can also be explicitly rebound. If a package was marked invalid because a trigger was dropped, the rebound package will no longer invoke the trigger.

A package that is in an inoperative state must be explicitly rebound before it can be used. Refer to the *Application Development Guide* for more information about binding and rebinding packages.

Federated database objects have similar dependencies. For example, dropping a server will make any packages or cached dynamic SQL referencing nicknames associated with that server invalid.

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, refer to the DROP statement in the *SQL Reference* manual.

Chapter 5. Administering DB2 Using GUI Tools

DB2 Universal Database provides Graphical User Interface (GUI) tools to help you administer local and remote databases easily from one central location called the “Control Center”.

This chapter presents an overview of the DB2 Universal Database administration tools that are available to you and explains how you can use them to get your job done easily and efficiently. It also gives you a summary of the Java Control Center and how you can customize the Control Center to include your own Java-enabled tools.

This chapter provides information on:

Administration Tools

The tools for administering DB2 are part of the Administration Client, a selectable component with each of the DB2 Universal Database products. The Administration Client is also available on a set of CD-ROMs that include the Administration Clients for all the operating systems on which DB2 is available. They allow you to install and use the Administration Client on any workstation: It does not matter whether your database servers are local or remote, or what operating system the database servers are running on. The tools enable you to perform the same functions from a Graphical User Interface as you could from the Command Line Processor. These functions include the entering of DB2 commands, SQL statements, or system commands. With the tools, however, you do not have to remember complex statements or commands and you get additional assistance.

The following tools are available from the Control Center toolbar:

- The Control Center. The Control Center is the main DB2 graphical tool for administering your database. From the Control Center, you get a clear overview of all the systems and database objects that are cataloged locally.
- The Satellite Administration Center. The Satellite Administration Center allows you to administer DB2 Satellite servers.
- The Command Center. The Command Center enables you to issue DB2 database commands, SQL statements, and operating system commands; recall previous commands; and scroll through access plans for SQL queries.
- The Script Center. The Script Center enables you to create, run and schedule operating-system-level commands and DB2 command scripts.
- The Alert Center. The Alert Center notifies you when thresholds that you have set have been exceeded or when a node in a multinode environment is no longer responding.
- The Journal. The Journal allows you to view the status of jobs and to view the recovery history log and messages log.
- The Information Center. The Information Center gives you quick access to the information in the DB2 product manuals and sample programs and provides access to other sources of DB2 information on the Web.
- The License Center. The License Center displays the status of your license as well as allowing you to configure your system for proper license monitoring.

For some functions that you can perform with the GUI tools, you are given the option of using a SmartGuide. SmartGuides are invoked from the pop-up menus in the Control Center. They provide a greater level of help by prompting you step-by-step on how to fill in the information necessary for the task you are doing and even making calculations and recommendations based

on information you supply. SmartGuides are very useful if you are a new database administrator or someone who only administers a database occasionally.

In DB2 Universal Database, the following SmartGuides exist:

- Backup Database. This asks you basic questions about the data in the database, the availability of the database, and recoverability requirements. It then suggests a backup plan, creates the job script, and schedules it. To invoke the Backup Database SmartGuide, select the icon representing the database you want to backup, click mouse button 2, and select **Backup –> Database using SmartGuide**.
- Create Database. This SmartGuide allows you to create a database, assign storage, and select basic performance options. To invoke the Create Database SmartGuide, select the Databases icon in the Object Tree pane, click mouse button 2, and select **Create –> Database using SmartGuide**.
- Create Table. This SmartGuide helps you to design columns using predefined column templates, create a primary key for the table, and assign the table to one or more table spaces. To invoke the SmartGuide, select the Tables icon, click mouse button 2, and select **Create –> Table using SmartGuide**.
- Create Table space. This SmartGuide lets you create a new table space and set basic storage performance options. To invoke it, select the Table Space icon, click mouse button 2, and select **Create –> Table space using SmartGuide**.
- Index SmartGuide. Use the Index SmartGuide to determine which indexes to create or drop for a given set of SQL statements. The recommendations are based on the workload that you specify. To invoke the Index SmartGuide, select the Indexes folder, click mouse button 2, and select **Create –> Index using SmartGuide**.
- Performance Configuration. This SmartGuide helps you tune databases by requesting information about the database, its data, and the purpose of the system. It then recommends new configuration parameters for the database and instance and automatically applies them if you wish. To invoke this SmartGuide, select the icon for a database, click mouse button 2, and select **Configure using SmartGuide**.
- Restore Database. This SmartGuide walks you through the process of recovering a database. To invoke the SmartGuide, select the icon for a database, click mouse button 2, and select **Restore –> Database using SmartGuide**.
- Configure Multisite Update SmartGuide. This SmartGuide lets you configure databases to enable applications to update multiple sites simultaneously when it is important that the data at all the sites must be consistent. To invoke this SmartGuide, select an instance, click mouse button 2, and select **Multisite Update –> Configure using SmartGuide**.

Note: SmartGuides do not exist for the DB2 for OS/390 subsystem.

Besides the graphical tools that you can invoke from the Control Center toolbar, there are some additional GUI tools that are not invoked directly from the Control Center toolbar.

- Performance Monitor. Performance Monitor is a tool to monitor DB2 objects such as instances, databases, tables, table spaces, and connections. You use this tool to detect performance problems and tune databases for optimum performance. The Performance Monitor is invoked as a selection on the pop-up menus in the Control Center.
- Event Monitor. Event monitor is a tool that lets you analyze resource usage by recording the state of the database at the time specific events occur. An Event Monitor is created by typing `db2emcrt` from a DB2 command line.
- Event Analyzer. Event Analyzer is a tool that allows you to analyze the data collected by the Event Monitor. An Event Analyzer is invoked by typing `db2evmon` from a DB2 command line.
- Visual explain function. The visual explain function lets you view the access plan for SQL statements as a graph so that you can tune your SQL queries for better performance. Prior to Version 6, you used the Visual Explain tool to view the access plans. In Version 6, Visual Explain is no longer a separate tool; however, the function is available on pop-up menus from various database objects in the Control Center, and also from the Command Center.

In addition to these tools, another useful tool for database administration that is not part of the Control Center is the:

- Client Configuration Assistant. The Client Configuration Assistant is a SmartGuide with the primary function of setting up communications from remote clients to servers.

All these tools are described in greater detail later on. The following section gives an overview of features found in the tools.

Common Tool Features

The following features are available in several tools:

- Show SQL and Show Command
- Show Related
- Help

Show SQL and Show Command

If a tool generates SQL statements, then the **Show SQL** pushbutton will be available on the tool interface. Similarly, a tool that generates DB2 commands will have a **Show Command** pushbutton available. Clicking one of these pushbuttons allows you to:

- See the SQL statements or DB2 commands that the tool generates based on the choices you made in the graphical interface. This information helps you to understand how the interface is working.
- Save the statements or commands as a script for future reuse. This capability saves you from having to retype the SQL statements or DB2 commands if you want to run the same statements or commands again. Once the SQL statements or DB2 commands have been saved in a script, you can schedule the script, or edit the script to make changes, or create similar scripts without having to retype the statements or commands.

To show the SQL statements or DB2 commands:

1. From the Control Center, go to a window or notebook for working with an object.
2. Click on the **Show SQL** or **Show Command** push button. The appropriate window opens.

Saving SQL statements and DB2 commands is particularly helpful if the SQL statements or DB2 commands are complex.

When you use the **Show Command** or **Show SQL** feature, you can either create new scripts which you can later edit, or you can close the dialog box to return to the original dialog to make changes. If you click the Create Script pushbutton, the New Command Script window appears. There you can edit the SQL statements or the DB2 commands before saving the script.

Show Related

Show Related shows the immediate relationship between tables, indexes, views, aliases, triggers, table spaces, User Defined Functions, and User Defined Types. For example, if you select a table and you choose to show related views, you only see any views that are based directly on the table. You would not see any views that are based on the related views because those views were not created directly from the table.

Showing related objects helps you to:

- Understand the structure of the database.
- Determine what indexes already exist for a table.
- Determine what objects are stored in a table space.

- Know what other objects are related to an object and are therefore affected by any actions you may take. For example, if you want to drop a table with dependent views, **Show related** shows you which views will become inoperative.

To use the Show Related feature:

- From the Control Center, select an object from the Contents pane and click mouse button 2.
- Select **Show Related**
- Click on the tab to open the page for the related objects you want. Different related objects are listed depending on the tab you select. Only objects that are directly related to the object that you have selected are shown.

  You can click mouse button 2 on a related object on the selected Page and select Show Related from the pop-up menu. The selected Page changes to show the objects related to your latest selection. You can also click on the down arrow next to the selected object to display a list of objects you previously selected to show relationships.
- Click **Close** to close the Show Related notebook and return to the Control Center.

Generate DDL

The **Generate DDL** function allows you to re-create and save in a script file the DDL and SQL statements and statistics of:

- Database objects
- Authorization statements
- Table spaces, nodegroups and buffer pools
- Database statistics

This allows you to:

- Save the DDL to create identically defined tables, databases, and indexes in another database, for example, for a database warehouse application
- Use the DDL to copy a database from the test environment to a production environment or from one system to another
- Edit the DDL to create similar objects

Clicking the **Generate DDL** pushbutton, brings up the Show Command window with statements generated by a utility know as the **db2look**. From the Show Command window, you can click the **Save Script** pushbutton to save the statements. The statements are put into a script. If you click the **Generate** button, the Run Script window opens.

You can select whether you want to generate DDL statements for selected schemas or all schemas within the database. You can then edit the script if you want to make changes before you use the script in a production

environment. To create identical databases using the generated DDL statements, you would simply use the script which you generated and run it in the new environment.

To generate DDL statements:

1. Highlight the object for which you want to generate DDL statements, and click mouse button 2.
2. Select **Generate DDL**. The Run script window appears.
3. Type a user ID and password, and click **OK**. A job is created with the contents of the **db2look** command. A DB2 message window appears with the job ID of the new job.
4. Click on **OK** to close the message window.
5. Use the Job History page of the Journal notebook to view the results of the job and to view the contents of a saved script associated with the job.
6. Select the job and click mouse button 2. Select **Show Results** from the pop-up menu. The Job Results window opens. The output of the **db2look** command is shown in the Job Output pane.
7. Select **Create Script** to create a script of the results. The New Command Script window appears.
8. Save the new script if you want to use it again.

Filter

In the Control Center, you can filter information that is displayed in the Contents pane, or you can filter information that is retrieved from a table as an actual result set. You can limit the number of objects that are displayed or the number of objects that are returned by creating filters for one or more objects. Once you have set the filter, you need to clear or delete the filter if you want to display all the objects in the tree once again.

Filtering the Display

To reduce the number of objects that appear in the Contents pane for more manageable administration:

1. Select the Filter icon from the Contents pane toolbar, located at the bottom of Control Center, or select Filter from the View menu bar.
2. Select the criteria to be used to reduce the number of objects.
3. Select the Enable filter checkbox to activate the filter.

When you later select an object to view its contents, the filter you have associated with the object limits the view according to the criteria you set earlier.

Filtering Retrieved Data

To reduce the number of rows returned in a query and improve the response time, you can define the output, or the result set that shows in the Contents pane when selecting an object.

1. Select a folder object from the tree and click mouse button 2.
2. From the pop-up menu, select **Filter**. The Filter window opens.
3. Use the Filter function to define a set of criteria for retrieving rows belonging to that object.

Defining a Filter to Retrieve a Specific Set of Data

To define a filter to retrieve a specific set of data:

1. From the Control Center, expand either the Databases or Subsystems folders depending on your platform.
2. Select an object for which you want to define the filter. Click mouse button 2 on that object.
3. Select **Filter** from the popup menu. This opens the Filter notebook.
4. On the Locate page, specify the name or other descriptive filter criteria of the selected object. The result of the filter is the results set associated with the selected object shown in the Contents pane of the Control Center.
5. On the Locate page, select a radio button to specify whether to meet all the conditions selected in the fields on the Locate page or to meet at least one condition.
6. On the Advanced page of the Filter notebook, you can use additional criteria by editing the text that is shown to further limit the number of returned rows.
7. Click **OK** to use the filter criteria you defined.

To automatically invoke this filter notebook based on numbers of rows, select Tools from the menu bar, and select Tools Settings from the popup menu. The **Select filtering when numbers of row exceeds** checkbox allows you to predefine a threshold of returned rows from any selection. When the threshold is reached, the Filter notebook appears so that you can limit the current retrieval based on the defined criteria. This is especially useful when a table has grown unexpectedly and was previously unfiltered. Depending on your platform, and your data, you could be attempting to return millions of rows, when you need only a subset of rows.

Help

Extensive help information is provided with the administration tools. A help button exists on all dialog boxes as well as on the menu toolbar. You can get general help as well as help on how to fill out the fields and perform tasks.

From the help menus, you can also access the index of terms or the reference information and the information provided in the product manuals.

The Control Center

Use the Control Center as your main point of administration to manage systems, DB2 instances, databases, database objects, such as tables, views, and user groups. You can also use the Control Center to access DB2 for OS/390 subsystems. All DB2 databases must be catalogued before they appear in the Control Center. The Figure 21 shows the primary features of the Control Center. Because of operating system differences, the Control Center on your system may appear different from the diagram.

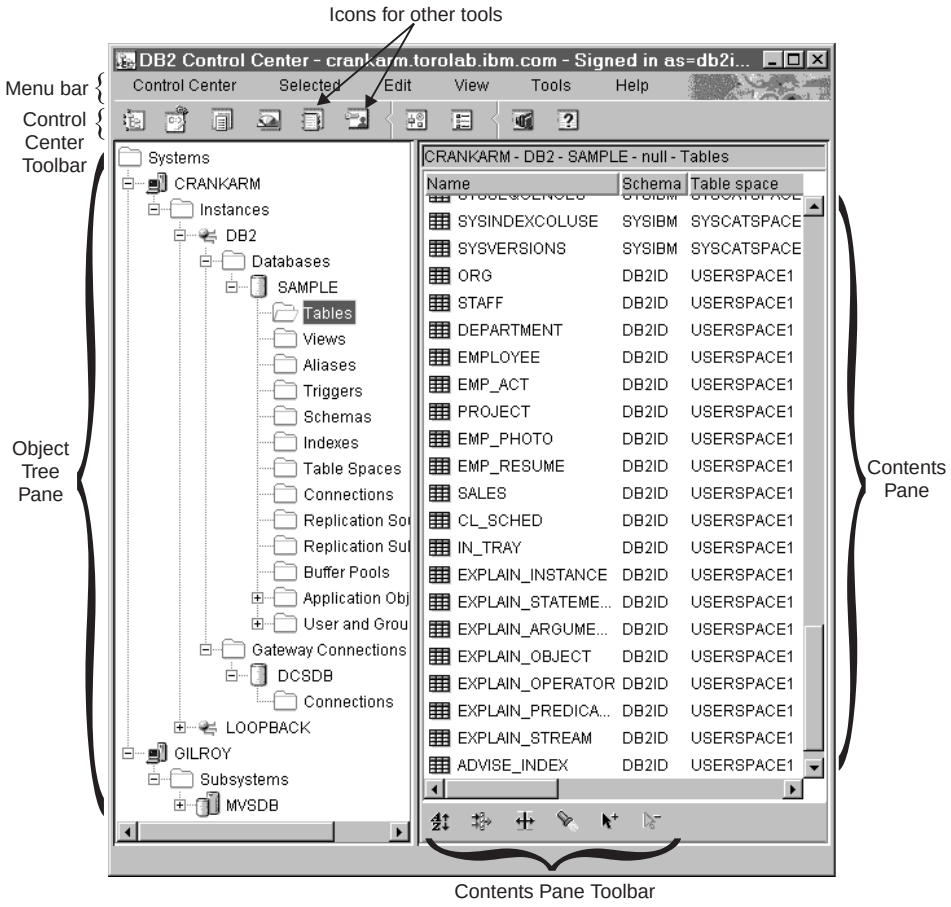


Figure 21. Control Center Features

Main Elements of the Control Center

The main elements of the Control Center are:

- Menu Bar. The menu bar is at the top of the screen. Selecting a menu from the menu bar allows you to perform many functions, such as shutting down the DB2 tools, accessing the graphical tools, and getting access to online help and product information. You should familiarize yourself with these functions by clicking on each item on the menu bar.
- Control Center Toolbar. Icons for the Control Center and the other tools are located on the Control Center toolbar. Hover help identifies each icon when the cursor is placed over the icon.

  You can change the settings for these tools by selecting the Tools Settings icon from the Control Center toolbar.
- Object Tree. The Object Tree is located on the left pane of the screen. It displays icons for all the database servers and objects that you can manage from the Control Center. You must first catalog a remote database server before it appears in the Object Tree pane. Local databases are automatically catalogued. Some objects in the Object Tree pane contain other objects. A plus sign (+) to the left of the object indicates that the object is collapsed. You can expand it by clicking the plus sign. A minus sign (−) appears to the left of an object when it has been expanded. To collapse the object, click the minus sign.
- Contents Pane. The Contents pane is located on the right pane of the screen. This pane shows all the objects that are contained in the selected object in the Object Tree pane, for example, if you select the tables folder in the Object Tree pane, all your tables show up in the Contents pane. If you select the databases folder, the Contents pane changes to show all databases. You can filter the columns that appear in the Contents pane by clicking the Filter icon in the Contents pane toolbar and specifying the required information or you can filter objects by selecting **Tools** on the toolbar and then **Tools Settings**. You must ensure that the **Enable Filter** check box is selected in the Contents Pane filter dialog.
- Contents Pane Toolbar. This toolbar appears at the bottom of the Contents pane. It allows you to tailor the information in the Contents pane. This toolbar is a common control which appears at the bottom or to the side of most detailed views throughout the product.

Using a Customized Control Center in DB2 for OS/390

Use the Customized Control Center on the DB2 for OS/390 platform as your own defined point of administration to manage subsystems, databases, or database objects, such as tables, views, and database users. You can use this Customized Control Center to access any DB2 for OS/390 objects that you define.

The main elements of the Customized Control Center are the same as those listed for the default Control Center. The Customized Control Center allows you to specify objects that you want to include in a personalized Control Center. This user-defined tree can be saved and invoked to administer DB2 objects. It does not replace the Control Center tree which is the default for all users but is useful if you want to access a set of objects in the same way each time the Control Center is invoked. You can create as many customized trees as you need, and each one can contain a different set of objects, which can be ordered in any way that you choose.

Using a customized tree reduces the effort of navigating through a fixed hierarchy of DB2 objects, and provides a method of grouping related objects. For example, you can define a tree that contains only tables with payroll information.

Systems That Can Be Administered

The Object Tree in the Control Center shows all the systems that are cataloged on the instance *to which you are currently attached.* If you attach to another instance, the Control Center shows the systems that are cataloged on that new instance. From the Control Center, you can administer database objects for the Universal Database family of products for OS/2, Windows, and UNIX platforms.

You can also administer DB2 for OS/390 subsystems from the Control Center if an Administration Server (DAS) is running on the DB2 for OS/390 system and if a DB2 Connect product is available to the client on which the Control Center is running. This requires either DB2 Connect Personal Edition to be installed on the client or DB2 Connect Enterprise Edition to be installed on a LAN and be accessible to the client. In addition to having a DB2 Connect product installed, a connection must be defined on the DB2 for OS/390 subsystem so that it can used by the Control Center.

From the Control Center, you can also replicate data between the DB2 Universal Database family of products, DB2 for AS/400, DB2 for VSE and VM systems, and DB2 for OS/390.

Objects that can be Administered

If you want to administer objects from the Control Center, you must add them to the object tree. If you remove a database, or uncatalog it outside of the Control Center, and you want to use the Control Center to perform tasks on it, you must add it to the object tree.

The DB2 Universal Database objects that you can administer from the Control Center are:

- Systems

- Instances
- Tables
- Views
- Indexes
- Triggers
- User-defined types
- User-defined functions
- Packages
- Aliases
- Replication Objects
- Users and Groups

The DB2 for OS/390 Version 5 objects that you can administer from the Control Center are:

- Buffer Pools
- Views
- Catalog Tables
- Storage Groups
- Aliases
- Synonyms
- DB2 Users
- Locations
- Application Objects (Collections, Packages, Plans, Procedures)
- Databases
- Tables
- Table Spaces
- Indexes
- Replication Source
- Replication Subscriptions

For Version 6 of DB2 UDB for OS/390, the following objects were added:

- Schemas
- Triggers
- User Defined Functions
- Distinct Types

To see what actions can be performed on each of these objects, select the object in the Object pane and click mouse button 2. A pop-up window appears listing the functions.

Displaying Systems in the Control Center

To display all of the systems that are cataloged on your system and which have DB2 installed:

1. Expand the Object Tree by clicking the plus sign (+) beside **Systems**. Icons representing the local machine and any remote machines are displayed. Your local system is represented by the Local icon. It only appears if the local machine is a DB2 server. If you click mouse button 2 on the Local icon, one of the options in the pop-up menu is called **Attach to administration server**. The Administration Server lets you take advantage of functions such as performance monitoring and scheduling. It is used as a service by the DB2 administration tools to satisfy operating system requests and it is automatically created and started for you. The default name for the Administration Server is DB2DAS00.
2. Expand the Local icon. The instance of DB2 on the local machine is displayed in a tree structure.

   On OS/2, Windows and supported DB2 UNIX-based systems, you can think of each copy of the database manager code as a separate *instance*, that is stored in a directory on your machine. On DB2 for OS/390, an instance is referred to as a subsystem. A default local instance is created when you install DB2. You can have several instances on a single system. You can use these instances to separate the development environment from the production environment, or to restrict sensitive information to a particular group of people. You can also tune an instance for a particular environment.
3. Expand the **Instances** icon. For each database that exists, an icon and the name are displayed.

Managing DB2 for OS/390 Objects

Using the Control Center, you can perform many of the functions of the existing DB2 for OS/390 Version 5 and DB2 UDB for OS/390 Version 6 products, such as creating, altering, and dropping objects, as well as run utilities that reorganize or load your data. However, before you can administer a DB2 for OS/390 subsystem from the Control Center, you must first add it to the object tree by configuring a connection to it.

Adding DB2 for OS/390 Subsystems

If you have the Client Configuration Assistant installed, you can use it to configure a connection to a DB2 for OS/390 subsystem easily. If you do not have the Client Configuration Assistant installed, you will have to configure the connection to the DB2 for OS/390 subsystem manually.

You use the Client Configuration Assistant to search the network for all the DB2 for OS/390 subsystems which are available on the LAN to your client. If

you would like to add one of the DB2 for OS/390 subsystems, you can use the **Add Database** SmartGuide to add the subsystem, or you can import a connection by using an access profile, or you can add the connection manually.

If you choose to search the network, you need to have a DB2 Connect product on your network with a connection defined for the subsystem. If you choose to use an access profile, you need to import a profile for the subsystem you want to add. If you need to manually configure the connection, you need to know the subsystem name, the communication protocol, and the communication protocol parameters such as the host name and the port number for TCP/IP, or the Symbolic Destination Name for SNA. Once you add the DB2 for OS/390 subsystem, you also get the objects for the gateway connections.

When you add a DB2 for OS/390 subsystem, it appears in its own section of the Control Center object tree separate from the DB2 Universal database objects. To see the DB2 for OS/390 V5 and DB2 UDB for V6 objects that reside in a particular subsystem, expand the object tree from the DB2 for OS/390 system icon that represents your DB2 for OS/390 system.

To see the list of actions that you can perform on a particular object, select the object as it appears in the object tree and click mouse button 2. A pop-up menu appears and shows the available actions you can perform on that object. For example, you can create, alter, or drop a view, as well as see its contents, modify the privileges on it, and show a list of other objects that are related to it. See the online help for the DB2 for OS/390 objects for more information on what functions you can perform.

Managing Gateway Connections

When a DB2 Connect server is installed, a Gateway Connections folder is displayed in the Control Center object tree under the instance object of the local system. The Gateway Connections folder contains a hierarchy of objects used to manage connections to host and AS/400 databases that are cataloged locally. The actions associated with these connection management objects can be used to list, force, and monitor host and AS/400 database connections.

The object tree under the Gateway Connections folder is used for managing connections to host and AS/400 databases but not for database administration tasks. However, if you need to add, change or remove a host or AS/400 database on the local system, you would use the Client Configuration Assistant and not the Gateway Connections tree.

Functions You Can Perform from the Control Center

From the Control Center, you can:

- Manage database objects. You can create, alter, and drop databases, table spaces, tables, views, indexes, triggers, schemas. You can also manage users.
- Manage data. You can load, import, export, reorganize data, and gather statistics.
- Schedule jobs. Jobs may be pending, running or completed executions of scripts. You can schedule jobs to start at particular times.
- Perform preventive maintenance by backing up and restoring databases.
- Monitor performance and perform troubleshooting.
- Replicate data.
- Configure and tune instances and databases.
- Manage database connections, such as gateways and subsystems. Manage applications.
- Analyze queries using Explain SQL to look at access plans.
- Launch other tools. For example, you can launch the Satellite Administration Center or the Command Center.

To see all the actions that you can perform on an object, simply select the object from the Object Tree pane or the Contents pane and click mouse button 2. A pop-up menu appears showing all the functions that you can perform on that type of object; for example, if you select the tables folder, you can create a new table with or without the help of a SmartGuide; monitor the performance of tables; filter which tables appear in the Contents pane and so on. The functions you can perform are different, depending on the object you select.

Click mouse button 2 on the objects in the Contents pane to perform additional functions on a specific object. For example, if you select one of your tables in the Contents pane and click mouse button 2, a pop-up window displays functions you can use on that table.

Creating New Objects

To create new objects:

1. Expand the databases folder. Object types are displayed as folder icons.
2. Click mouse button 2 on the folder icon for an object, for example, click on the **Tables** icon. The pop-up menu is displayed. For some objects, you get two options to perform a function. One option is to use the SmartGuide. SmartGuides do not exist for all functions that you can perform.
3. Select **Create**. Since there is a SmartGuide to create a table, you get two options, one of which is to create the table using the SmartGuide. If you choose the SmartGuide option, you are prompted for information and given suggestions on what choices you should make. The SmartGuide is especially useful for new users or people who create databases infrequently.

Working with Existing Objects

When you click an object such as the table folder in the Object Tree pane, all tables already existing appear in the Contents pane. You can then select a table you want to work with and click mouse button 2 to invoke any functions that you wish to perform on that specific table.

For more information about using the Control Center, go to its online help, available from its **Help** menu or by pressing F1 anywhere in the Control Center.

Locating objects (DB2 for OS/390 only)

You can search for a database or subsystem object easily by using the Locate notebook. This allows you to:

- Find an object without having to navigate through the tree structure of the Control Center. The object could be in a database or subsystem, table space, or across databases and tables and supporting objects.
- Locate objects (table spaces, tables, and indexes) across multiple databases within a subsystem.

Use the Locate page of the Locate notebook to specify the search criteria. Use the Advanced page of the Locate notebook to further customize the search. Edit the text provided on the Advanced page and add or modify the search criteria.

To locate an object defined within a database or a DB2 for OS/390 subsystem:

1. From the Control Center, click mouse button 2 on an object. Select **Locate** from the popup menu. The Locate notebook opens.
2. From the Object type field, select the type of database object to search. The list of target objects available varies, depending on the object from which you begin your search.
3. On the Locate page, fill in the search criteria. You must type in at least one search criterion and you can use wild cards to help in the search. Characters are folded to uppercase unless you use valid delimiters to enclose lower case characters or the extended character set.
4. On the Locate page, select a radio button to specify whether to meet all the conditions selected in the fields on the Locate page or meet at least one of the conditions.
5. Click **OK** to use the search criteria. The results of your search are displayed in the Locate Result window. The format of the output table depends on the type of object for which you searched.
6. To repeat the search with the same or different criteria, click **APPLY**.

7. You can select a row that appears in the Locate Result window and right click on that row to see a popup menu with additional actions that you can perform.

The Satellite Administration Center

The Satellite Administration Center is a set of tools that is available from the DB2 Control Center. They allow you to set up and administer collections of DB2 servers from a central point. Each DB2 server that belongs to a group is known as a satellite. Administering satellites from a central point means that DB2 can be hidden from anyone using a DB2 satellite, thereby avoiding the need for them to learn about database administration.

Use groups to organize DB2 servers that have shared characteristics, such as the applications that run on them or the database configuration that supports the application. The DB2 servers are similar in terms of their database configuration, usage, and purpose.

By grouping the DB2 servers together, you can administer groups of DB2 servers rather than having to administer each DB2 server individually. If you acquire additional DB2 servers to serve the same function as the DB2 servers of an existing group, you can add them to that group by using the Satellite Administration Center.

From the Satellite Administration Center, you can create groups, satellites, application versions, batches, and authentication. You can also define success code sets and perform other functions associated with the administration of the satellite environment. Information about the satellite environment is stored in a central database known as the satellite control database. This database records, among other things, which satellites are in the environment, the group each satellite belongs to, and which version of the end-user application a satellite is running. This database is on a DB2 server that is known as the DB2 control server.

Before the functionality of the Satellite Administration Center can be enabled, you must first catalog a satellite control database (SATCTLDB) on the Control Center. When it is enabled, you can use the Satellite Control Center to set up and maintain satellites, groups, and the batches that the satellites execute when they synchronize for their application version.

To set up and maintain its database configuration, each satellite connects to the satellite control database to download the batches that correspond to its version of the end-user application. The satellite executes these batches locally, then reports the results back to the satellite control database. This process of downloading batches, executing them, then reporting the results of the batch

execution is known as synchronization. A satellite synchronizes to maintain its consistency with the other satellites that belong to its group and are running the same version of the end-user application.

The Command Center

You can start the Command Center from the Control Center by clicking the Command Center icon on the toolbar.

The Command Center lets you:

- See the resulting output of one or many SQL statements and DB2 commands in a result window. You can scroll through the results and generate a report.
- Create, and save command scripts to the Script Center. You can edit the command script to create new scripts. From the Script Center, the command script can then be scheduled to run as a job at any time that you specify.
- Run SQL statements, DB2 commands and operating system commands. When you run operating DB2 commands from the Command Center, you do not have to precede the command by **DB2**. You can run operating system commands in any supported operating system script language, such as REXX, by preceding them with an exclamation mark (!). Using the Command Center to run the commands and statements allows you to issue many commands at once, without the need to type and run each command individually.
- Get quick access to the DB2 administration tools, such as the Control Center, from the main toolbar.
- See the access plan and statistics associated with an SQL statement before execution.

The Script Center

You can start the Script Center by selecting the icon from the Control Center Toolbar. The Script Center is a tool that allows you to create scripts by writing a set of commands and statements, which you can schedule to run whenever you require. You can import scripts that you created earlier or scripts that you saved in the Command Center. You can select scripts from the set of scripts saved and you can edit existing scripts to create new scripts, copy scripts, or remove scripts.

You can edit a script inside the Script Center or outside the Script Center using your own editor. If you run a script from within the Script Center, you get the added advantage of having the results logged in the Journal.

To run operating system commands from a script in the Script Center:

1. Select **Script –> New**. The New Command Script window opens.
2. For **Script Type**, select the **OS command** radio button.
3. Enter the script name, description, and working directory.
4. Enter the commands.
5. Click on **OK**.

From the Script Center, you can view information, such as the description and script type about all command scripts that are known to the system, and you can perform the following tasks:

- Create a command script that contains DB2 and operating system commands
- Run a saved command script immediately
- Schedule a script to run at a later date or at a regular interval; for example, you might want to create a script that collects statistics for several tables. You could then schedule the job to run overnight. You can schedule jobs by running them unattended at scheduled intervals by specifying the hours, days, weeks, months, multiple times a week, or multiple times a month you want to run the job. A job is created whenever you schedule a script or run a script immediately.
- Access the Journal from the toolbar to see the jobs that use a particular script and to see the status of all scheduled jobs
- Edit a saved command script

Using an Existing Script with the Script Center

To use the Script Center with pre-existing scripts which you did not create from the Script Center:

1. From the **Control Center** toolbar, click on the **Script Center** icon. The Script Center opens.
2. Select **Script –> Import**. The File Browser window opens.
3. Select an existing script file and click on **OK**. The New Command Script window opens. The script is displayed in the lower part of the window which is a script editor. Complete the **Instance**, **Script name**, **Script description** and **Working directory** fields, and select a **Script type**.
4. Click on **OK**. The script will be created in the Script Center.

Scheduling a Saved Command Script to Run

To schedule a script:

1. Click the **Script Center** icon on the Control Center toolbar. The Script Center opens.

2. Click with mouse button 2 on the script you want to schedule to run and select **Schedule** from the pop-up menu. The Scheduler window opens.
3. Select the frequency for the job, and a completion action, such as a completion message or another command script to be launched.
4. Click **OK**. This starts a pending job that you can track in the Journal.

The Journal

You can start the Journal by selecting the icon from the Control Center toolbar. The Journal allows you to monitor jobs and review results. From the Journal, you can also display the recovery history and DB2 messages. The Journal allows you to monitor pending jobs, running jobs, and job histories; review results; display recovery history and alert messages; and show the log of DB2 messages.

Working with Jobs

Use the Journal to work with jobs. To open the Journal:

1. Click the **Journal** icon from the Script Center toolbar. The Journal opens.
2. To see the jobs which are scheduled to be run at a later time, click the **Pending jobs** push button. You see your job in the list of pending jobs. You also see all the information about the jobs. You can perform actions on a pending job, such as reschedule it, show the scripts associated with it, or run it immediately. When a saved script is modified, all jobs that are dependent on it inherit the new modified behavior.

From the Journal, you can also see the jobs that are currently running and the job histories.

The other pages in the Journal window are:

- The Recovery page. This page displays the recovery history (the details from backup, restore operations, and load operations) and lets you restore the recovery log.
- Alerts page. This page shows all alerts.
- The Messages page. This shows all messages issued through the DB2 administration tools.

The online help for the Journal provides detailed steps for working with jobs and logs.

The License Center

You use the License Center to display license status and usage information for DB2 products installed on your system. You can also use the License Center to configure your system for proper license monitoring. The License Center allows you to:

- Add a new license.
- Upgrade from a trial license of the product to an authorized license.
- View the details of your license.

If you view the details of the license information, you see the:

- Product name
- Version information
- Expiry date
- Registered users
- Concurrent users
- Number of entitled users
- Concurrent number of users
- Enforcement policy

The Alert Center

The Alert Center is the tool that monitors your system to warn you about potential problems. You can set the Alert Center to automatically open to display any monitored objects that have exceeded their threshold and are therefore in a state of alarm or warning. You set up the thresholds using the Performance Monitor, which is invoked from the Control Center. The color of the icon indicates the severity of the warning. A red icon indicates an alarm. A yellow icon indicates a warning. The data returned for the performance variable is displayed. See the online help for instructions on how to analyze the data.

Client Configuration Assistant

The Client Configuration Assistant (CCA) is a SmartGuide whose primary function is to set up communications easily from a remote client to a database server saving you from having to configure parameters manually; however, the Client Configuration Assistant is also an excellent tool to add databases and new systems by allowing you to discover and get access to all available systems on your network. When you use the Client Configuration Assistant, you do not need to know the location of the server nor do you need to manually configure the server, because the Client Configuration Assistant can

search the network for systems, instances, and databases and then use this information to configure communications.

The Client Configuration Assistant:

- Requests information about protocols, ports, and associated network information that is required for configuration.
- Configures database connections.
- Allows you to update or delete existing database connections and display existing configurations.
- Provides lists of all databases you are connected to.
- Searches the network for DB2 databases.
- Imports and exports database connections. This allows you to use connections that exist on other machines in your network and use them to connect to the same systems easily without having to know all the configuration information.

Searching for Databases

With the Client Configuration Assistant, you can:

- View local or remote systems. This gives you a picture of the network to which you are connected.
- See any Universal Database server that has an Administration Server running. If you find a DB2 Connect server, you can discover and catalog any DB2 for OS/390 subsystems that can be accessed through that server. This capability allows you to discover and configure databases directly rather than by using the Control Center.
- Import and export database connections using profiles. If you want to copy database connections that are on another machine or if you want to make a template client to distribute to other systems, you can take a snapshot of your configuration using the CCA and export the snapshot. You might use this, for example, if your machine has a connection that you want replicated on several machines. When you open the CCA, there are two buttons: **Import** and **Export**. To copy the database connections configured, you export the connections. This produces a `.profile` file which you can then send to others who would import using the **Import** button or **Add Database** from the main window of the CCA. You can also import server profiles from the Control Center. From the Control Center, click **Export** and you would then use that profile as a source to import. To copy the database connections configured, you export the connections.

You can still add databases manually if you know that they exist and you do not have a gateway server running on them.

Performance Monitor

The Performance Monitor provides information about the state of DB2 Universal Database and the data that it controls. It is a graphical utility that can be customized for your database environment. You can define thresholds or zones that trigger warnings or alarms when the values being collected by the Performance Monitor are not within acceptable ranges.

You can monitor DB2 objects such as instances, databases, tables, table spaces, and connections by selecting the object in the Object Tree pane or in the Contents pane and clicking mouse button 2. From there, you can choose to start monitoring activity.

When an object is being monitored, the color of the icon appears green, yellow, or red to indicate the status of the monitor. The colors represent the severity of the problems as defined by the thresholds which you have set. Green signifies that the monitor is running and everything is fine. Yellow is a warning and signifies that the monitor is reaching the thresholds that you have set. Red indicates an alarm and that the monitor has reached the threshold. You can use the predefined monitors that are included with DB2 or you can create your own monitors.

To see what information the Performance Monitor is collecting, click mouse button 2 on the object and select **Show Monitor Activity** in the pop-up window.

Use the information from the Performance Monitor to:

- Detect performance problems
- Tune databases for optimum performance
- Analyze performance trends
- Analyze the performance of database applications
- Prevent problems from occurring

The Performance monitor lets you analyze trends by creating a visual presentation of database information such as disk activity, buffer pool usage, amount of prefetch, lock usage, and record blocking at specific intervals.

You use the tool when you need to monitor an existing problem or when you want to observe the performance of your system. It lets you take a snapshot of database activity and performance data at a point in time. These snapshots are used for comparison over time. Each point on the graph represents a data value. The steps for taking snapshots are provided in “Monitoring Performance at a Point in Time” on page 261. This information can help you to identify and analyze potential problems, or identify exception conditions

which are based on thresholds that you set. Use the performance tool if you need to know the performance of the database manager and its database applications at a single point in time and look at trends over time. Use it also to get a visual overview of what elements are in a state of alarm. This helps you to identify which parameters may need tuning. You can then look closely at the parameters that have been set for that element and change it to improve performance.

Event Monitor

In contrast to taking a point in time snapshot, an event monitor collects information on database activities over a period of time. This collected information provides a good summary of the activity for a particular database event, for example, a database connection or an SQL statement. Event monitoring records the state of the database at the time specific events occur. It allows you to obtain a trace of the activity on the database. Event monitor records are stored and then analyzed after the data has been captured. Use the event monitor when you need to know how long a transaction took or, for example, how much CPU an SQL statement used. You then use the Event Analyzer to read the data recorded from the event monitor.

For each database connection, there is one connection event record produced. For each statement run in that connect, a statement record is produced. Each connection event record maps to one row in the Connections View window of the Event Analyzer. This window shows information for each application that connected during the monitored period, including:

- Application name
- Execution ID
- Connect time
- Total CPU time
- Lock wait time
- Total sort time
- Deadlocks
- Disconnect time
- Application ID

Each statement event record maps to one row in the Statements View window in the Event Analyzer.

Using the Monitor Tools

The Performance Monitor and the Event Analyzer provide the following benefits:

- Comprehensive, flexible data collection. Over 200 performance variables are supported including buffer pool and I/O, lock and deadlock, sorting, communication, agent, and logging information. Data is shown for database managers, databases, table spaces, tables, buffer pools, connections, transactions, and SQL statements.
- Easy-to-use, intuitive viewing. Data can be viewed in real time using easy-to-read graphs or textual views conveniently organized into logical groups. Both details and summary views are provided, with the ability to access more detailed information.
- Robust alert capabilities. For any performance measurement, you can define exception conditions by specifying a threshold value. The threshold values are used to visually identify when a performance measurement reaches or exceeds the threshold value by plotting a measurement in a particular zone on the performance graph. When the threshold value is reached, you can specify that you want any or all of the following actions to occur.
  - You are notified through the Alert Center.
  - You receive an audible alarm.
  - A program is run.
  - A message is displayed.
  - No notification is given.

Figure 22 on page 260 illustrates how the monitors work together.

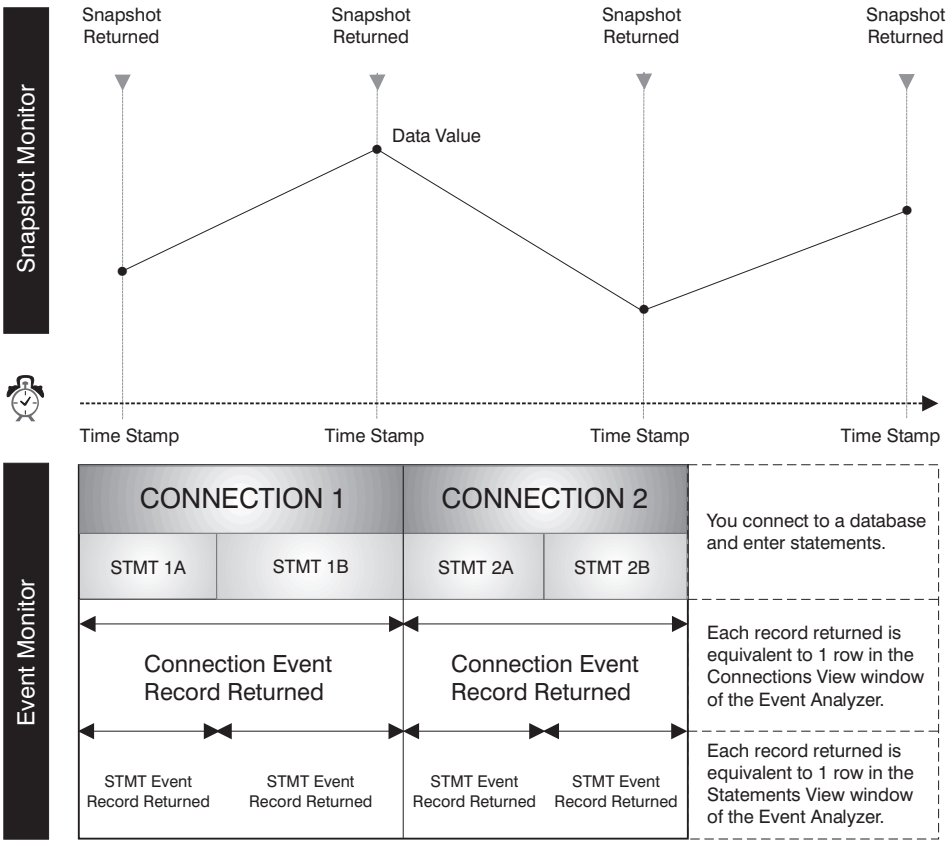


Figure 22. Comparison: Getting Snapshots and monitoring Events. (Event monitor, Event Analyzer)

Considerations for Monitoring and Tuning a Database

Before you start monitoring and tuning your database, you should do the following

- Define your objectives. For example, you may want to understand how applications use resources at the instance level at a specific point in time so that you can, for example, check if database concurrency is reduced when a special application is started. Or you may want to understand which instance-level events occur when an application is running, for example, if there is poor overall performance when a particular application is running.
- Determine what information you will analyze. For example, to see if bottlenecks are hardware related, you may want to take snapshots to monitor database connection activity or table space, buffer pool, and I/O activity. To see if the bottlenecks are environment-related, you would use the Event Analyzer to monitor if:
  - Too many database tasks are scheduled during peak time

  - There is a high number of user connections
  - Database partitioning (hardware load balancing) is not well optimized
  - The server is being used for more than just a database server

  Some of the visible effects are, for example:
  - Queries / responses are slow
  - Scheduled tasks are not completing on time
  - Applications are timing out
- Decide whether you will use the predefined monitors that are available with DB2 or whether you will create your own monitors.

The next section describes how to take snapshots and how to use the Alert Center to keep track of any performance-related problems.

Monitoring Performance at a Point in Time

If you want to do complex data collection and analyze the data to pinpoint potential problems, use the Performance Monitor to take snapshots of your system and watch performance data change over time.

The tools lets you:
- Graph performance information
- Define performance variables
- Set the capture frequency of performance snapshots
- View the results of performance calculations
- Define threshold values and threshold actions
- Generate and store alerts
- View summary information (for example, all databases)

The following types of information are captured:
- Information about long-lived activities (such as database activity when an application is taking too long to complete).
- Counters that keep track of information about the current level of activity (such as the number of open cursors for a database).
- Cumulative information about database activity (such as the maximum number of connections made while a database instance is active, or the total number of SQL statements executed against a particular database).

Taking snapshots at predefined intervals provides a picture of the current state of the activity in the database manager and its applications. This information can be used to:
- Detect performance problems

- Analyze performance trends
- Tune database manager and database configuration parameters
- Analyze the performance of database applications

Performance information is available for the following database objects:

- Instances
- Databases
- Tables
- Table spaces
- Database connections

For each, a variety of performance variables can be monitored. The Performance Variable Reference Help, available from the **Help** menu of any Snapshot Monitor window, provides a description of all the performance variables. These variables are organized into categories. By default, all performance variables are monitored, but the categories can be turned on and off through the administration tools. The following categories have been set on by default:

- Instance: Agents, Connections, Sort
- Database: Lock and Deadlock, Buffer Pool and I/O, Connections, Lock and Deadlock, Sort, SQL Statement Activity
- Table: Table
- Table space: Buffer Pool and I/O
- Database Connections: Buffer Pool and I/O, Lock and Deadlock, Sort, SQL Cursors, SQL Statement Activity

From the Control Center, you can only capture snapshots from one instance of a database manager at a time. This means that the API that is used to get snapshot information is issued only once for all monitored database objects in a database manager. This decreases the overhead on a database manager.

For detailed information on how to generate snapshots, see the online help.

Predefined Monitors

The DB2 Performance Monitor contains a set of predefined monitors, which you can use as they are or which you can copy and modify to meet your requirements. They provide a comprehensive set of performance calculations. You cannot change the name, equation, or text description of an IBM-supplied performance monitor; however, you can change the threshold values and the alert actions. Use the predefined monitors to learn about performance monitoring and to create your own monitors by copying a predefined monitor and adding or removing performance variables from your copy.

The Predefined Monitors that are supplied with DB2 are:

- Monitoring Capacity. Use this monitor to get information on system capacity. This monitor can be checked on a regular basis to see the overall usage of your system over time.
- Sort. Use this monitor to ensure that your sort heap and sort heap threshold parameters are set correctly. This monitor should be run when you first start your system, in peak periods of activity, or as applications change.
- Locking. Use this monitor to determine how much locking is occurring in your system, and whether your lock list parameters are set appropriately.
- Cache. Use this monitor to optimize cache usage. By monitoring these values during peak periods, you can determine if you need to increase the size of the cache.
- Bufferpool. Use this monitor on small tables to determine whether they require their own buffer pools.
- Deadlocks. Use this monitor to determine whether your applications are getting into deadlocks.
- Fast Communication Manager. Use this monitor to see the percentage of memory used to transfer information between nodes.
- Prefetchers. Use this monitor to determine whether you have enough prefetchers defined for the system.
- Disk Performance. Use this monitor to watch input and output. This monitor contains performance variables that focus on disk performance at the database and table space levels.
- Global Memory. Use this monitor to watch application memory use.
- Long Running Memory. Use this monitor to help determine why a query is taking a long time to complete.
- Gateway Connections. Use this monitor to watch gateway connections.

For examples of how to use a predefined monitor, see the online help provided for performance monitoring.

To see a list of available monitors, from the Control Center, click mouse button 2 on the **Systems** folder, and select **List Monitors** from the pop-up menu. The List Monitors window opens. It lists the monitors that are stored on the JDBC server to which you are currently connected. For each monitor, you see the name of the monitor, a description, the status, whether it is the default monitor, and who created the monitor. The status of the monitors indicates the status of the monitors on the local system, and not on the JDBC server. The Default for level indicates the default monitor at the instance, database, table, table space or connections level. For the predefined monitors, the Created by column contains **NULLID**. The right side of the window contains pushbuttons which allow you to perform various tasks on the monitors.

You can choose which monitor is started as the default monitor for an object.

Once you have started a performance monitor, you can click on the Alert Center button on the toolbar to see the status of any objects that you are monitoring and which are in a state of alert because they have reached any of their threshold values. They appear only for the period of time during which the threshold is exceeded.

If you want to keep a close watch on the objects being monitored, you can keep the Alert Center open or you can keep the **Show Monitor** window open on the summary page and look for any red or yellow entries. You can also modify the Control Center settings so the Alert Center opens automatically if a new warning or alarm is added to it. From the Alerts Center, you can also temporarily suspend the alerts while monitoring continues.

Action Required When an Object Appears in the Alert Center

You can set the Alert Center to open automatically to display any monitored objects that are in a state of alarm or warning (that is, their thresholds have been exceeded). You can change this default from the Tools Settings window.

If you see an object in the Alert Center, click mouse button 2 on the object and select **Performance Monitor –> Show Monitor** to view the performance details for that database object.

See the online help available from the **Help** menu of any Performance Monitor window for instructions on how to analyze the data.

Analyzing an Event for a Period of Time

The Event Analyzer is another DB2 performance tool. Use this tool when you want diagnostic information for an event that has taken place. You use the Event Analyzer in conjunction with an event monitor. For example, you can use an event monitor to trace database activity, such as connections, transactions, statements, and deadlocks, while a database is active. An event monitor can also record cumulative performance data that is logged when an application disconnects from the database. After the event monitor has created the event monitor file, you look at your performance information using the Event Analyzer.

The event monitor tools let you perform the following:

- Create event monitors to monitor the types of database events that are of interest to you.
- Activate an event monitor to start collecting event data. The data is stored in a file.
- Stop an event monitor from collecting event data.

- View the trace-type summary information that is produced by the event monitor.
- Remove an event monitor when you no longer have a need for it. You are also given the option to clean up its trace files.
- Display a list of event monitors associated with the database.
- View the definition of an event monitor.

The Event Analyzer lets you view the data generated by an event monitor for the following event types:

- Database connection activity (the period of time between a connection and its disconnection)
- Transactions (units of work)
- SQL statement executions
- Detection of deadlock activity

Event Analyzer

You can create an event monitor for the following event types and then use the Event Analyzer to view the collected information: however, use the `db2evmon` executable (described in the *Command Reference* and the *System Monitor Guide and Reference*) to view data generated for the:

- Deadlocks
- Database activity
- Table space activity
- Table activity
- Statement activity

To analyze event data using an event monitor and the Event Analyzer, follow the steps below. They represent only one example of how to create an event monitor for connection and statement events. To create an event monitor:

1. From a command line in the Command Center, type **db2emcrt**. The Event Monitor window opens.
2. Click on **Event Monitor** and choose Create from the menu. The Create Event Monitor window opens.
3. In the field, specify a name for the event monitor you are creating. This new event monitor cannot have the same name as any existing monitor. Blank spaces are not allowed in the name.
4. In the Enterprise - Extended Edition product only, select a node where the event monitor files will reside from the `On Node` drop down list.
5. In the Enterprise - Extended Edition product only, select a scope for the event monitor. By default, the scope is Global.

6. Select one or more of the check boxes to indicate the type of events that you want to monitor. Note that the Deadlocks event type is the default selection.
7. Indicate when you want this monitor to start. Note that `Start now` is the default selection.
8. Define one or more conditions for connections, statements, or transactions that will control monitoring at these levels.
9. Identify a path (directory name) where the monitor will write the event data files.
10. Click on **Options** to open a window for **Specifying Event Monitor File** options. These options determine how monitor output is handled and can affect the performance of your event monitor.
11. Click on OK to create the monitor, or Cancel to exit without creating a monitor.
12. Turn off the event monitoring, by clicking mouse button 2 on an event monitor and select **Stop Event Monitoring** from the pop-up menu.

    This forces the event monitor to write the trace file. If the monitor is not turned off, information is only written to disk when the buffer is full or all connections end. From the Event Monitors window, you can view the resultant event data by clicking mouse button 2 on the event monitor you created, and selecting **View Event Monitor Files** from the pop-up menu. The Monitored Periods View window opens.

To access the event data from the Event Analyzer:

1. From a command line in the Command Center, type **db2eva** to start the Event Analyzer. The Event Analyzer window opens.
2. In the Path field, identify the path (directory name) where the data files are stored. If the files have not been moved, this will be the path that was specified when the event monitor was created. If the files were moved, then specify that directory. You can click on ... to list existing directories.

   Note: If data files are stored remotely, you must FTP the files to your local machine in order to view them. Depending on file size, this transfer could take some time. Files can be transferred to any local path. It is not necessary to choose the same path that was used when they were created.
3. Click **Ok** to access the data files contained in the directory, or Cancel to exit. The Monitored Periods View window opens.
4. Click mouse button 2 on a monitored period, and select **Open as –> Connections** from the pop-up menu. The Connections View window opens. This shows the list of connections that were made during the event monitoring session. (There may be more than one connection listed. The connection you are interested in may not be the first one in the list.)

5. Click mouse button 2 on a connection, and select **Open as –> Statements** from the pop-up menu. The SQL Statements View window opens. It displays all statements for the selected connection. Columns of information are provided for each statement, including:
   - Operation
   - Package name
   - Creator
   - Start time
   - Elapsed time
   - Total CPU time
   - Text

The online help for the event monitor and the Event Analyzer provide detailed instructions for creating event monitors and viewing the resultant event data.

Analyzing SQL Statements

You can view the access plan for explained SQL statements as a graph and use this information to tune your SQL queries for better performance.

An access plan graph shows details of:

- Tables (and their associated columns) and indexes
- Operators (such as table scans, sorts, and joins)
- Table spaces and functions

Prior to Version 6, you would use a tool called Visual Explain to view the access plans. In Version 6, you can no longer invoke Visual Explain as a separate tool from the command line, however, you can still invoke the visual explain *function* from various database objects in the Control Center and from the Command Center. In this section, the term *visual explain function* is used for this capability.

You use the visual explain function 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, the visual explain function 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).

Improving Performance of a Query

You use the visual explain function to analyze and tune SQL statements. It presents a graphical view of the access plan for explained SQL statements. 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. You can use the information available from this graph to find ways to tune your SQL queries for better performance.

The visual explain function captures information about how SQL statements are compiled. This information allows you to understand the plan and potential execution performance of SQL statements.

This information can help you:

- Design application programs.
- Design databases.
- Understand how two tables are joined: the join method, the order in which the tables are joined, the occurrence of sorts and type of sorts.
- Determine ways of improving the performance of SQL statements (for example, by creating a new index).
- 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 re-binding the package might improve performance. It also helps you determine whether collecting statistics might improve performance.
- Determine whether or not an index was used to access a table. If an index was not used, the visual explain function can help you determine which columns could be included in an index to help improve query performance.
- View the effects of performing various tuning techniques for the purpose of better performance 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.

Analyzing a Simple Dynamic SQL Statement

This section provides a simple example of how to get started analyzing a dynamic SQL query.

1. From the Control Center, click mouse button 2 on the SAMPLE database, and select **Explain SQL** from the pop-up menu. The Explain SQL Statement window opens.

2. In the **SQL text** field, enter the following SQL statement:

   ```
   select * from staff order by name
   ```

3. Click **OK**. The Access Plan Graph window opens. The graph represents the path that the optimizer chose as the most efficient in order to provide the results for your query.
4. Optional: Double-click any of the nodes (for example, the RETURN operator node). The Operator Details window opens, showing the details for that operator.

The explained SQL statement is saved automatically. To view it later:

1. From the Control Center, click mouse button 2 on the SAMPLE database, and select **Show explained statements history** from the pop-up menu. The Explained Statements History window opens.
2. Locate the entry you want. You can look at the **SQL text** column to see the SQL statement you had previously explained.
3. Click mouse button 2 on the entry, and select **Show access plan** from the pop-up menu. The Access Plan Graph window opens.

The online help for Visual Explain (accessible from the **Help** menu) provides details on how to interpret the Access Plan Graph window in order to improve the performance of SQL statements. The online help also contains detailed examples to help you learn how to use Visual Explain.

Managing Remote Databases

The following section shows you how to:

- Add a remote system
- Add the instance you want to work with for that system
- Add the database you want to work with under that instance

DB2 first checks in the node directory (which contains an entry for all servers to which a database client can connect and the communications protocol used in the connection) to see if the remote system is already known. If the remote system is not known, with a system, instance, or database on a remote system, you need to set yourself up as a client to the remote system.

After you install DB2, you can use the Client Configuration Assistant to search the network for systems, instances, and databases and configure communications for them. You then add the remote system by cataloging it. This creates an entry for the system in the node directory so that its instances and databases can be made known. Next, you must add the instances and databases for the system by cataloging them to create an entry for them in the node directory and database directory, respectively. This creates an entry for

them in the node directory and database directory, respectively). When the configuration is complete, the remote systems are displayed in the Control Center so that you can work with them.

To add a remote system:

1. From the Control Center, click mouse button 2 on the **Systems** object and select **Add**. The Add System window opens.
2. Enter the system name in the **System name** field.

   If the **Discover** configuration parameter for the instance is set to **search** and the **discover comm** configuration parameter is not blank, you can select **Refresh** to get a list of the remote systems. You can then select one of the systems from the list below the **System name** field.
3. Type the remote instance name in the **Remote instance name** field.
4. Select the type of operating system for the remote system from the **Operating system** list.
5. Select the protocol you want used for communications with the remote locations. For a local system, **Local** is automatically selected and is the only valid protocol. For the remote systems the possible protocols are:
   - APPC
   - IPX/SPX
   - NetBIOS
   - TCP/IP
   - Named pipe (on Windows NT and Windows 9x operating systems only)

   Only the protocols that the computer is currently set up for appear in the listbox.
6. Enter the appropriate protocol parameters.
7. Enter a comment to be associated with the system.
8. Click **Apply** to add the system to the node directory.

Next, add the instance you want to work with on that system:

1. From the Control Center, click mouse button 2 on the **Instances** object belonging to the system you just added.
2. Select **Add**. The Add Instance window opens.
3. Enter the required values in the fields.
4. Click the **Refresh** push button to have a list of existing instances displayed.
5. Select the instance you want to work with.
6. Click the **Apply** push button, then the **Close** push button.

Finally, add the database you want to work with under that instance:

1. From the Control Center, click mouse button 2 on the **Databases** object.
2. Click **Add**. The Add Database window opens.
3. Enter the database name, type of communication protocol, and, optionally, an alias. An alias in this case is an alternative name used to identify a database.
4. Click the **Refresh** push button to display a list of existing databases for that instance.
5. Select a database.
6. Click the **Apply** push button, then the **Close** push button.

Managing Users

As a database administrator, you might need to control the type of access people have to data, or restrict their view of the data. The following information tells you how to use the administration tools to manage database authorities and privileges for database objects.

Database *authorities* involve actions on a database as a whole. When a database is created, some authorities are automatically granted to anyone who accesses the database. For example, CONNECT, CREATETAB, BINDADD and IMPLICIT_SCHEMA authorities are granted to all users. Database *privileges* involve actions on specific objects within the database. When a database is created, some privileges are automatically granted to anyone who accesses the database. For example, SELECT privilege is granted on catalog views and EXECUTE and BIND privilege on each successfully bound utility is granted to all users.

Together, privileges and authorities act to control access to an instance and its database objects. Users can access only those objects for which they have the appropriate authorization, that is, the required privilege or authority.

Granting and Revoking Authorities and Privileges

You can use the DB2 administration tools to grant and revoke privileges for users and groups for databases, table spaces, tables, views, and schemas.

1. From the Control Center, click mouse button 2 on the database, table, view, schema or index for which you want to grant or revoke privileges. Select **Authorities** or **Privileges** from the pop-up menu. The Authorities window or Privileges window opens.
2. Select the **User** page to work with user authorities or privileges or the **Group** page to work with group authorities or privileges.
3. Select one or more users or groups. To add a user or group to the list, click the **Add User** or **Add Group** push button.

4. Along the bottom of the window, select **Yes**, **No**, or **Grant** for each individual authority or privilege. **Grant** is displayed only for objects for which it is a valid option.
5. When you have finished, click the **Apply** push button.

If you want to review or change the objects that a particular user is authorized to, you can select a user, and click mouse button 2, then add or change authorization to an object or remove authorization.

Moving Data

DB2 provides the import and load utilities to help you move data into a table from existing sources. The information provided in this section is a brief overview of moving data. For more detailed information on moving data, you should refer to the *Data Movement Utilities Guide and Reference* manual.

The import utility takes data from an input file and inserts it into a table or view. In this case, the input file contains data that was extracted from an existing source of data, such as a Lotus 1–2–3 file or an ASCII file. You can also use the import utility to re-create a table or view that was saved by using the export utility. The following information tells you how to import data.

Once you have an input file available in a supported format, use the Import notebook to insert data from the file into an existing table. If this table already contains data, you can either replace or append to the existing data with the data in the file.

You can also use the Import notebook to create a new table that is populated by an input file, or delete existing rows in the selected table and repopulate it using data from the input file.

To import a file into an existing table:

1. Open the File page of the Import notebook.
2. Optional. Specify the Import notebook.
3. Optional. Retrieve Large objects.
4. Optional. Specify column import options.
5. Click **OK**

To open the File page of the Import notebook:

1. From the Control Center, expand the object tree until you find the **Tables** folder.
2. Click the **Tables** folder. Any existing tables are displayed in the contents pane.

3. Click mouse button 2 on a table in the contents pane and select **Import** from the pop-up menu. The Import notebook opens with the File page displayed.

To specify the file options:

1. In the **Import file** field of the File page, enter the name of the file that contains the data you want to import.
2. Specify the type of file to import by selecting one of the following
   - Non-delimited ASCII format (ASC)

     Non-delimited ASCII data is data that is aligned in columns.
   - Delimited ASCII format (DEL)

     Delimited ASCII data is a commonly used way of storing data that separates column values with a user-defined delimiting character, such as a comma.
   - Worksheet format (WSF)
   - Integrated exchange format (IXF)

     PC/IXF is a structured description of a database table or view. Data that was exported in PC/IXF format can be imported or loaded into another DB2 Universal Database product database.

   See the online help for the specific products and releases that are supported.
3. Optional: Specify file type modifiers by clicking the corresponding **Options** push button. The Options window for that format opens.
4. Select an **Import mode**. The available import modes vary depending on the file type you selected.
5. Optional: In the **Commit records** field, enter the number of records to import before the changes are committed.
6. Optional: In the **Restart** field, enter the number of records in the file to skip before beginning the import action.
7. Optional: In the **Compound** field, type a number to specify how many SQL statements will be executed (in an executable block).
8. Optional: Select the **Insert an implied decimal point on decimal data (IMPLIEDDECIMALPOINT)** check box.
9. In the **Message file** field, type the name of the file that will contain warning and error messages that occur during import.

To retrieve large objects from separate files, use the Large Objects page of the Import notebook to retrieve large objects (LOBs) from the path or paths that store the LOB files:

1. Click the **Retrieve large objects (LOBs) in separate files (LOBSINFILE)** check box to enable the options on the Large Objects page.

2. Specify the location of separate LOB files in the LOB paths list box by clicking the **Add** push button. These paths are searched (in the order in which they appear in the **LOB paths** list box) for the LOB files specified in the LOB column of the input file.
3. Click **OK** to accept the defaults on the other notebook pages and begin the import process.

Specify column import options. Use the **Columns** page of the **Import** notebook to specify column import options:

1. Click one of the radio buttons in the **Include columns by** box to specify the column method that will be used to import data file columns into the table. The available methods vary depending on the file type and mode you selected on the File page.
2. Optional: Specify or change the import file column attributes by clicking the **Change** push button.

   This option is not available if you selected the **Default (method D)** radio button.

Managing Storage

As a database administrator, you need to estimate the size of tables and indexes, and to check the amount of space available in a table space adding more space to an existing table space when it gets full.

This section describes how to:

- Estimate the size of tables and indexes
- Check the amount of space available in a table space
- Add more space to an existing table space when it starts to get full

Estimating Table and Index Size

You can estimate the amount of storage space required for new or existing tables or indexes by invoking the Estimate Size dialog. Invoke this dialog by selecting individual tables and indexes and clicking mouse button 2 on them, or select **Estimate Size** from the Create Table and Create Index windows. The size is estimated on the definition of the particular table and its dependent indexes. The estimate is the projected amount of storage space that would be used when the table has a given number of rows. The minimum and maximum space is also estimated based on the smallest and largest size of variable length fields. When invoked on a table or an index, the **Estimate Size** dialog is prefilled with the specifications of the table, and contains numbers relating to the table and all of its dependent indexes. When you click the

Refresh pushbutton, the estimated size, minimum size, and maximum size are updated based on the number you enter in the **New total number of rows** and **New average row length** fields.

Estimating the size of a table or index is helpful if you want to:

- Create a new table and you want to know how large to make the table space.
- Create a new table based on the size estimate of an existing table.
- Know how much space is used by different table and index objects in a table space because the system is running out of storage space.
- Estimate the projected size of a table prior to loading data.

Note: When you use Estimate Size on an Enterprise-Extended Edition product, the size estimates are based on the logical size of the data in the table and not on the database partition.

If you have not updated the statistics for the table for some time, you can click the **Run statistics** pushbutton to update the statistics for the selected table. If you select an index and then press the **Run statistics** button, the statistics are run on the related table.

To estimate the size for a table:

- Open the Estimate Size window.
- Select a different value for **New total number of rows** or accept the default.
- Click **Refresh** to view the size estimates for the new value.
- Select a different value for **New average row length** or accept the default.
- Click **Refresh** to view the size estimates for the new value.

Checking Space Available in a Table Space

To check the amount of space available in a DMS table space:

1. From the Control Center, double-click on the **Table Spaces**. A list of all the table spaces appears in the contents pane.
2. Scroll to the columns entitled **Allocated size**, **Size used** and **Percentage used** to see details related to the amount of space available in a table space. Space is measured in pages where one page is 4 KB.

You can customize the order of the columns and which columns are displayed by using the Customize Columns icon at the bottom of the Contents pane.

To check the amount of space available in an SMS table space, use the facilities provided by your operating system to monitor space usage and to ensure that available room in the directory for the table space is not exhausted.

Adding More Space to a Table Space

Capacity for a DMS table space is the total size of containers allocated to the table space. When a DMS table space reaches capacity (depending on the usage of the table space, 90% is a possible threshold), you should add more space to it. The database manager will automatically rebalance the tables in the DMS table space across all available containers. During rebalancing, data in the table space remains accessible.

For a DMS table space that has reached its capacity, you can add another container:

1. From the Control Center, click mouse button 2 on the table space in the Contents pane for which you want to add a container, and select **Alter** from the pop-up menu. The Alter Table Space window opens.
2. Click **Add**. The Add Container window opens.
3. Select the **File** or **Raw device** radio button, and complete the fields. See the online help for assistance.
4. Click **OK**.

In general, you cannot extend the size of an SMS table space very easily because SMS capacity depends on the space available in the file system and the maximum size of the file supported by the operating system. However, depending on your operating system, you may be able to increase the size of a file system using the operating system facilities. For an SMS table space on a UNIX-based system, you can increase the size of the table spaces by using the appropriate UNIX-based system commands. See the documentation for the UNIX-based system you are running. If the file system containing the SMS table space also contains non-DB2 files, you may be able to move these files to another file system, thus making more room available in the file system for DB2's use. You can also perform a redirected restore which involves restoring a table space into a larger number of containers than it was backed up from. You can perform a redirected restore from the Restore Database notebook: From the database you want to restore, select **Restore -> Database** from the pop-up menu.

Troubleshooting

DB2 provides a troubleshooting manual that is intended for technical support representatives for DB2 servers and clients. It helps you:

- Identify problems or errors in a concise manner
- Solve problems based on their symptoms
- Use available diagnostic tools
- Develop a troubleshooting strategy for your day-to-day DB2 operations.

The *Troubleshooting Guide* presents these basic troubleshooting topics:

- Good troubleshooting practices
- Troubleshooting on the server
- Troubleshooting on the client
- Troubleshooting host communications
- Troubleshooting applications
- Troubleshooting and problem determination.

The *Troubleshooting Guide* presents these advanced troubleshooting topics:

- The DB2 process model
- Using logged information
- Taking traces
- Diagnostic tools for UNIX-based, OS/2, and Microsoft and Windows operating systems.

Up-to-date bulletins and technical documentation are available from the World Wide Web at http://www.software.ibm.com/data/db2/library/.

See the section at the end of this book for the details on how to contact IBM.

Replicating Data

Replication is the process of taking changes stored in the database log at the source server and applying them to the target server. You can use replication to define, synchronize, automate, and manage copy operations for data across your enterprise. You can automatically deliver the data from a host system to target sites. For example, you can copy data and applications to branch offices, retail outlets, and even sales representatives' laptops.

The two operational components in replication are Capture and Apply. The Capture component captures changes made to data in source tables which have been defined for replication by reading the database log. The Apply component reads the changed data previously captured and stored in a change data table and applies it to the target tables.

Using the Control Center, you can do the setup required for replication using the **Define as replication source** and **Define subscription** actions. The replication components Capture and Apply run outside the DB2 administration tools.

Replication administrators can perform the following actions from the Control Center:

- Define replication sources
- Define replication subscriptions
- Create control tables and target tables
- Specify SQL to enhance data during the apply process

The high-level steps for replicating data are as follows. Refer to the *Replication Guide and Reference* for details.

1. Design a replication scenario (map the source and target tables).
2. Define a replication source (this relates to the capture action).

To define a replication source:

1. Specify source columns to capture.
2. Choose replication options.
3. Define a replication subscription (this relates to the Apply action).
4. Alter the source table with the Data Capture Changes option.
5. Start Capture to read and store data changes.
6. Start Apply to replicate changes to target tables.

To define a replication subscription:

1. Name the subscription set.
2. Specify the database and target table.
3. Specify the target columns.
4. Specify the row selection.
5. Specify SQL for run-time processing.
6. Set the subscription timing.

Using Lightweight Directory Access Protocol

The DB2 Version 6 Control center can be used as a central control point, if it is running on a system where an LDAP client is installed. All the database instances registered on the LDAP server will be cataloged automatically on the client. They will show up in Control Center as regular nodes on the navigator tree. These databases can be managed the same way as the other databases that you have cataloged on your machine (except the ADD DATABASE option is not yet available in this release). Of course, the local DB2 system needs to be set-up with the same network protocols (for example, TCP/IP, NETBIOS, or IPX/SPX) in order to talk to the databases on other database servers.

To administer an LDAP database, select the database and click mouse button 2. A pop-up window lists the functions which you can perform. For more

information on LDAP, see “Appendix N. Lightweight Directory Access Protocol (LDAP) Directory Services” on page 829.

Using a Java Control Center

You can run the Control Center as a Java application or as a Java applet through a web server. In both cases, you need a supported Java Virtual Machine (JVM) installed on your machine to run the Control Center. To run the Control Center as a Java application, you must also have the correct Java Runtime Environment (JRE) installed. A Java Virtual Machine can be a Java Runtime Environment (JRE) for running applications or a Java-enabled browser for running applets.

Java **applications** are run just like other applications on your machine, provided you have the correct JRE installed. Java **applets** are programs that run within Java-enabled browsers. The Control Center applet code can reside on a remote machine and is served to the client’s browser through a web server. If you run the Control Center as a java applet, you must use a supported Java-enabled browser running on a Windows 32-bit or OS/2 operating system. Currently, there are no supported browsers for UNIX operating systems.

The Control Center JDBC Applet Server must be started with a user account that has administrator authority on the machine where the Applet Server resides. You can set your Control Center JDBC Applet Server to start automatically at startup time.

Running the Control Center as a Java Applet

To run the Control Center as a Java applet, you must have a Web server set up on the machine that contains the Control Center applet code and the Control Center JDBC Applet Server. The Web server must allow access to the `sqllib` directory. If you choose to use a virtual directory, substitute this directory for the home directory. For example, if you name your virtual directory `temp`, then you should use `sqllib/temp`. DB2 does not support the installation of the Control Center on a FAT drive for OS/2 because an OS/2 FAT drive does not support long filenames required by Java. For more information on installing and configuring the Control Center as a Java application or Java applet, see the *Control Center Readme*.

Using Your Java-based Tools for Administration

In Version 6, DB2 includes a set of Java interfaces that allow you to extend the capabilities of the Control Center. The Java interfaces allow you to:

- Add additional items to the menu list when working with objects.
- Add buttons to the Control Center toolbar.

To use this capability, you must have the right level of Java software installed. For more information on using this function, see the *What's New* manual.

Chapter 6. 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.
- 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.
- Monitoring access of data by users which is discussed in "Chapter 7. Auditing DB2 Activities" on page 333.

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 or *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

DB2 Administrators may now allow others to change passwords on AIX and Windows NT EEE systems through the profile registry variable DB2CHGPWD_EEE=<boolean>.

The default for this variable is NOT SET (disabled). Other values for DB2CHGPWD_EEE are the standard boolean values used by other DB2 profile variables.

The DB2 Administrator is responsible for ensuring that the passwords for all nodes are maintained centrally using either a Windows NT Domain Controller on Windows NT, or NIS on AIX.

Note: If the passwords are not maintained centrally, enabling the DB2CHGPWD_EEE variable will allow for the possibility that passwords may not be consistent across all nodes. That is, if a user uses the ″change password″ feature, then the user's password will only be changed at the node to which they connect.

DB2 UDB on AIX has added the functionality to log failed password attempts with the operating system and detect when a client has exceeded the number of allowable login tries as specified by the LOGINRETRIES parameter.

"Selecting an Authentication Method for Your Server" on page 287 provides additional information about the system entry validation checking that is particularly relevant if you have remote clients accessing the database.

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 (the exception to this general case being when privileges are granted to PUBLIC: these are considered when static SQL is processed). Specific cases where group membership does not apply are noted throughout DB2 documentation, where applicable.

"Privileges, Authorities, and Authorization" on page 305 presents further details on these topics.

Federated Database Authentication and Authorization Overview

Because a DB2 federated database system can access information in multiple DBMSs, additional steps might be required to secure your data.

When planning your authentication approach, consider the fact that users might need to pass authentication checks at data sources as well as at DB2. In a federated system, authentication can take place at DB2 client workstations, DB2 servers, data sources (DB2, DB2 for OS/390, other DRDA servers, Oracle), or a combination of DB2 (client or DB2 server) and data sources. Even in DCE environments, specific steps might be required if data sources

require a user ID and password. See “Federated Database Authentication Processing” on page 299 for more information.

Similarly, your users must pass authorization checking at data sources and at DB2. Each data source (DB2, Oracle, DB2 for OS/390, and so on) maintains the security of the objects under its control. When a user performs an operation against a nickname, that user must pass authorization checking for the table or view referenced by the nickname.

Selecting User IDs and Groups for Your Installation

Security issues are important to the DB2 Administrator from the moment the product is installed. The respective platform-specific *Quick Beginnings*, books present all of the information required to plan for, install, and configure DB2.

The steps to completing the installation of DB2 require a user name, a group name, and a password. During the installation, the administrator has default values for each of these requirements. Once the defaults have been used during the installation of DB2, the administrator is strongly recommended to create new user names, group names, and passwords before creating the instances where the databases will reside. Using new user names, group name, and passwords will minimize the risk of a user other than the administrator learning of the defaults and using them in an improper fashion within instances and databases.

Another security recommendation following the installation of DB2 is the changing of the default privileges granted to users. During the installation process, System Administration (SYSADM) privileges are granted by default to the following users on each operating system:

OS/2	A valid DB2 user ID which belongs to the UPM Administrator or Local Administrator group.
Windows 95	Any Windows 95 user.
Windows NT	A valid DB2 username which belongs to the Administrators group.
UNIX	A valid DB2 username which belongs to the primary group of the instance owner’s user ID.

SYSADM privileges are the most powerful set of privileges available within DB2. (Privileges are discussed later in this chapter.) As a result, you may not

want all of these users to have SYSADM privileges by default. DB2 provides the administrator with the ability to grant and revoke privileges to groups and individual user IDs.

The platform-specific information to create and assign groups and user IDs is found in the various *Quick Beginnings* books. By updating the database manager configuration parameter SYSADM_GROUP, the administrator can control which group is defined as the System Administrative group with System Administrator privileges. You must follow the guidelines below to complete the security requirements for both DB2 installation and the subsequent instance and database creation.

Any group defined as the System Administrative group (by updating SYSADM_GROUP) must exist. The name of this group should allow for easy identification as the group created for instance owners. User IDs and groups that belong to this group have system administrator authority for their respective instances.

You should consider creating an instance owner user ID that is easily recognized as being associated with a particular instance. This user ID should have as one of its groups, the name of the SYSADM group created above. Another recommendation is to only use this instance owner user ID as a member of the instance owner group and not to use it in any other group. This should control the proliferation of user IDs and groups that could modify the instance environment.

The created user ID should always be associated with a password to allow for authentication before entry into the data and databases within the instance. The recommendation when creating a password is to follow your organization's password naming guidelines.

On UNIX-based platforms, a group for fenced User Defined Functions (UDFs) and stored procedures must be created, and any user IDs that use fenced UDFs or stored procedures must be a member of this group. As with the SYSADM group, the name of the fenced UDFs or stored procedures group should allow for easy identification. User IDs that belong to the fenced UDFs or stored procedures have whatever authority and privileges that are associated with the group as their default.

For security reasons, we recommend you do not use the instance name as the Fenced ID. However, if you are not planning to use fenced UDFs or stored procedures, you can set the Fenced ID to the instance name instead of creating another user ID.

The recommendation is to create a user ID that will be recognized as being associated with this group. The user for fenced UDFs and stored procedures is

specified as a parameter of the instance creation script (db2icrt ... -u <FencedID>). This is not required if you install the DB2 Clients or the DB2 Software Developer's Kit.

There are rules for the naming of all objects and users. Some of these rules are specific to the platform you are working on. For example, there is a rule regarding the use of upper and lower case letters in a name.

- On UNIX platforms, names must be in lower case.
- On OS/2, names must be in upper case.
- On Windows platforms, names can be in upper, lower, and mixed-case.

See "Appendix D. Naming Rules" on page 691 for other naming rules.

The db2icrt command creates the main SQL library (sqllib) directory under the home directory of the instance owner.

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. Refer to "Configuring DB2" in *Administration Guide, Performance* 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.

If you intend to access data sources from a federated database, you must consider data source authentication processing and definitions for federated authentication types. See "Federated Database Authentication Processing" on page 299 for more information.

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.

SERVER_ENCRYPT

Specifies that the server accepts encrypted SERVER authentication schemes. If the client authentication is not specified, the client is authenticated using the method selected at the server.

If the client authentication is DCS or SERVER, the client is authenticated by passing the user ID and password to the server. If the client authentication is DCS_ENCRYPT or SERVER_ENCRYPT, the client is authenticated by passing a user ID and encrypted password.

If SERVER_ENCRYPT is specified at the client and SERVER is specified at the server, an error is returned because of the mismatch in the authentication levels.

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.

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. Refer to "Configuring DB2" in *Administration Guide, Performance* 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.

To protect against all clients except DRDA clients from DB2 for MVS and OS/390, DB2 for VM and VSE, and DB2 for OS/400, set the `trust_allclnts` parameter to DRDAONLY. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

The `trust_clntauth` parameter is used to determine where the above clients are authenticated: if `trust_clntauth` is ″client″, authentication takes place at the client. If `trust_clntauth` is ″server″, authentication takes place at the client when no password is provided and at the server when a password is provided.

Table 24. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations.

TRUST_ ALLCLNTS	TRUST_ CLNTAUTH	Untrusted non–DRDA Client Authen-tication no password	Untrusted non–DRDA Client Authen-tication with password	Trusted non–DRDA Client Authen-tication no password	Trusted non–DRDA Client Authen-tication with password	DRDA Client Authen-tication no password	DRDA Client Authen-tication with password
YES	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT
YES	SERVER	CLIENT	SERVER	CLIENT	SERVER	CLIENT	SERVER
NO	CLIENT	SERVER	SERVER	CLIENT	CLIENT	CLIENT	CLIENT
NO	SERVER	SERVER	SERVER	CLIENT	SERVER	CLIENT	SERVER
DRDAONLY	CLIENT	SERVER	SERVER	SERVER	SERVER	CLIENT	CLIENT
DRDAONLY	SERVER	SERVER	SERVER	SERVER	SERVER	CLIENT	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.

DCS_ENCRYPT
Specifies that DB2 Connect accepts encrypted SERVER authentication schemes. If the client authentication is not specified, the client is authenticated using the method selected at the server.

If the client authentication is DCS or SERVER, the client is authenticated by passing the user ID and password to DB2 Connect. If the client authentication is DCS_ENCRYPT or SERVER_ENCRYPT, the client is authenticated by passing a user ID and encrypted password.

If DCS_ENCRYPT is specified at the client and DCS is specified at the server, an error is returned because of the mismatch in the authentication levels.

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 293.

DCE_SERVER_ENCRYPT

Specifies that the server accepts DCE authentication or encrypted SERVER authentication schemes. If the client authentication is DCE or not specified, the client is authenticated using DCE Security Services. For more information on DCE Security, see "Using DCE Security Services to Authenticate Users" on page 293.

If the client authentication is SERVER or DCS, the client is authenticated by passing the user ID and password to the server. If the client authentication is SERVER_ENCRYPT or DCS_ENCRYPT, the client is authenticated by passing a user ID and encrypted password. The authentication type of the client cannot be specified as DCE_SERVER_ENCRYPT. If the authentication type of an instance is specified as DCE_SERVER_ENCRYPT, all local applications will use DCE as the authentication scheme. This also applies for any utility commands that do not require a database connection or an instance attachment.

In addition to allowing a mix of DCE and SERVER_ENCRYPT authentication types, the DCE_SERVER_ENCRYPT authentication type also alleviates one of the limitations when using groups within DCE. When the authentication type is set to DCE_SERVER_ENCRYPT, the assumption is that the group list being requested other than at authentication time, come from the base operating system and not from DCE. You, as the administrator, can then set up a user on the server to match the short DCE name in order to provide group list support outside that which is supported at authentication time.

Notes:

1. The type of authentication you choose is important only if you have remote database clients accessing the database or when you are using federated database functionality. Most users accessing the database through local clients are always authenticated on the same machine as the database. An exception can exist when DCE Security Services are used. For information about supporting and using remote clients, refer to your *Quick Beginnings* manual.
2. 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 fail-safe 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 fail-safe 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

3. See "Appendix J. How DB2 for Windows NT Works with Windows NT Security" on page 807 for additional information on Windows NT Security.

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.

Note: There are several vendor products that support DCE. To ensure that DB2 UDB for Windows NT can work with IBM's DCE product in the area of security services, two new DLLs have been provided: `db2dces.ibm` and `db2dcec.ibm`. (These DLL files are only appropriate for Windows NT.) If you purchase and use IBM's DCE product for security services, these two files must be copied to `db2dces.dll` and `db2dcec.dll` respectively. If you are considering another vendor's DCE product, you should contact the vendor service organization and the DB2 UDB service organization to discuss whether the vendor's DCE implementation for security services will work with DB2 UDB.

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}
```

```
{reserved no}
{scope {}}
{trigbind {}}
{trigtype 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.

  If a DCE group is to be used for a DCE principal, it must also have a mapping to a DB2 authid which has the proper authority such as SYSADM or SYSCTRL authority.

  Please note that the authorization identifier (authid) specified in the DCE schema used to map a DCE principal name to a DB2 authid *must* be specified in uppercase. Use of a lowercase or mixed case authid will result in an error.

How to Setup a DB2 Server to Use DCE

Servers must be registered principals 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:

```
db2 update database manager configuration using authentication DCE
 sysadm_group DCE_group_name
```

Then perform a `dce_login` to a valid DB2 DCE user who has SYSADM authority 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 294 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.

One of the functions of the DB2 Administration Server is to start DB2 instances. When AUTHENTICATION = DCE, the DCE principal used in the DB2 keytab file for the instance must have a valid DCE principal to DB2 authid mapping. This mapping is required for the DB2 Administration Server to start the DB2 instance. The valid mapping allows this ID to act as a client as well as a server.

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.

DCE authentication as performed by DB2 flows DCE Tickets obtained using the OSF DCE Generic Security Services Application Programming Interface (GSSAPI). As such, all authentication for DCE Security takes place at the database protocol layer. Certain communication mechanisms may provide additional communication layer security, which is not necessarily integrated with DCE. In cases where the communication layer authentication can be kept entirely independent of the database protocol layer authentication, no restrictions will be enforced. However, the criteria for both the database protocol layer and the communication layer authenticating must be satisfied before a connection can be successfully established. In cases where the database protocol layer and the communication protocol layer authentication mechanisms interact, their use may be restricted if some combinations result in a security exposure.

DCE authentication may be used in conjunction with TCPIP SOCKS support; however, the two security mechanisms work independent of one another. This may mean that not only must the user provide a valid DCE login context, but must also be logged on to a local operating system user ID that meets the criteria of the SOCKS Server.

DCE authentication may be used in conjunction with NT Named Pipes; however, the two security mechanisms work independent of one another. Not only must the user provide a valid DCE login context, but he must also be logged on to the NT Domain to a user ID that meets the criteria for the NT Named Pipes support.

In order to address possible confusion where DCE principals and local operating system user IDs are both used for authentication, as in the above two examples, an integrated DCE logon can be used. In this case, when logging on to a system, the user is automatically logged into the appropriate DCE principal as well. See the DCE documentation for your platform for details on how to use this feature, if it is supported. Note that in using this approach, the same name is used for the DCE principal and the local operating system ID. This may mean that the same value that is contained in the DCE encrypted ticket also flows on the wire unencrypted in the communication layer.

DCE authentication can only be used with APPC communications when the SECURITY parameter is set to NONE. This is to avoid the possibility of sending an unencrypted principal and/or password in the communication layer, while using an encrypted DCE token for the same principal in the database protocol layer. DCE Security at the APPC layer is not supported by DB2 at this time.

Federated Database Authentication Processing

If you have installed the distributed join installation feature and set the DBM configuration variable *federated* to 'YES', your DB2 system is operating as a federated system. Database authentication settings in a federated system differ slightly from standard DB2 definitions. More importantly, in a federated system you must consider the authentication requirements of your data sources. In general, data sources (DB2, Oracle, DB2 for OS/390, and so on) are set up to require authentication. That means you must ensure that IDs and passwords (as required) can flow to data sources. DB2 provides several methods for supporting authentication at data sources, all of which are explained in this section.

Authentication Settings

SERVER
Specifies that clients connecting to DB2 provide a user ID and password to access DB2. In this case a user ID and password are available for transmission to data sources. You control what is actually passed to the data sources through server options and user mappings, but authentication information is available for transmission to the data source.

CLIENT
Specifies that authentication takes place on the database partition where the application is invoked using operating system security. No passwords are available for transmission directly to data sources. In this case, if a data source requires authentication, you must create one

or more user mappings. You must also ensure that server options are set properly to transmit correct user ID and password information to the data source.

Exercise extreme caution when using CLIENT authentication. Consider this form of authentication only for secure networks. A user has SYSADM authority for the federated database when the following conditions are met:

- Authentication is set to CLIENT.
- The user has root status at the client.
- The user knows the SYSADM's authorization name.
- The user defines an authorization name on the client that is the same as the SYSADM's on DB2.

DCS Specifies that authentication takes place at a data source–not DB2. In this case, standard DB2 authentication processing is bypassed. User IDs and passwords are passed directly to data sources, depending on server option settings. Authentication takes place only at Oracle or DB2 Family data sources.

Exercise caution when authentication is set to DCS. Authentication is done at neither the client nor at DB2. Any user who knows the SYSADM authentication name can assume SYSADM authority for the federated server.

DCE If authentication is set to DCE, only a user ID is available for transmission to data sources. No password is available. If a data source requires authentication processing (user ID and password), you must define a user mapping that will transmit a password (and possibly a user ID) to the data source. If the data source trusts the DB2 connection, user mappings are not required because the ID received from the external security system can be passed to the data source.

Other DB2 authentication settings are possible, and one or more can result in the availability of a password at DB2 for transmission to data sources. If DB2 and client authentication settings result in the transmission of a password to DB2, that password is available for additional authentication processing at data sources. See Table 24 on page 290 for more information.

Passing IDs and Passwords to Data Sources

There are four ways to control the transmission of authentication information to data sources: DB2 authentication settings, user mappings, server options, and APPC security settings:

Authentication Settings

The purpose of this section is to clarify how authentication settings influence global authentication processing in a federated system (the definitions for authentication settings are in “Authentication Settings” on page 299). For example, if DB2 authentication is set to SERVER or DCS, a user ID and password are required for a connection. Therefore, a user ID and password are available for transmission to data sources. If authentication is set to DCE or CLIENT, and authentication is not taking place at the DB2 system containing the federated database, only a user ID is available. If data source authentication processing requires a password (or perhaps a different user ID and a password), you must create a user mapping. If authentication is set to CLIENT, and the *trust_clntauth* parameter setting is SERVER, it is possible that a password is sent to DB2 and that it is available for transmission to data sources.

User Mappings

DB2 can send either the authorization name used to connect to DB2 or an authorization name defined at DB2. User mappings store authorization names defined at DB2. They are created with the CREATE USER MAPPING statement.

User mappings are flexible: you can map an ID to a new ID and password or just a password. You can use them to provide missing information or to change an ID and password to values accepted at the data source.

To create or alter a user mapping, you must hold one of the SYSADM or DBADM authorities, or your authentication ID must match the authorization name specified for the statement.

An example of a user mapping statement is:

```
CREATE USER MAPPING FOR "SHAWN" SERVER DB21 OPTIONS (REMOTE_AUTHID "SHAWNBCA",
REMOTE_PASSWORD "MAPLELEAF")
```

where a DB2 authentication ID (SHAWN) is mapped to the remote ID SHAWNBCA and remote password MAPLELEAF for a server named DB21.

If the only difference between the authorization name (or password) at DB2 and the authorization name (or password) at the data source is the case of the passed string, consider using server options to fold the case to the desired setting instead of creating new IDs and passwords. See “Server Options” on page 302 for more information.

You must create a user mapping when your authentication setting is DCE and a data source requires authentication processing (a password is expected). DB2

will only pass the DCE user ID to data sources. A password must be mapped to that user ID and then sent to the data source.

Server Options

Server options can be used to provide overall authentication support. Use them to indicate if passwords are passed to data sources (typically yes) and whether user IDs and passwords need to be folded to uppercase or lowercase. Server options are set using the CREATE SERVER, ALTER SERVER, and SET SERVER OPTION statements.

Server options specific to authentication processing are discussed in the rest of this section. A more complete list of server options is in “Using Server Options to Help Define Data Sources and Facilitate Authentication Processing” on page 192.

Password Server Option: The default setting for password is ’Y’ (passwords are sent to data sources). Leave or set this option to ’Y’ for all cases where a data source will perform authentication and is not expecting an encrypted password.

DB2 can transmit encrypted passwords. Set the server option password to ’ENCRYPTION’ if passwords should be sent in an encrypted form to DB2 Family data sources. It is recommended that you set password to ’ENCRYPTION’ if your authentication setting at DB2 is DCS_ENCRYPT or SERVER_ENCRYPT.

A user ID is always sent to data sources.

ID and Password Folding Options: Authorization names and passwords, in some cases, might need to change. Different data sources can have different authorization name and password requirements (regarding the use of uppercase or lowercase) for IDs and passwords.

DB2 provides two server options that can help you resolve naming differences. The option names are **fold_id** and **fold_pw**, and their settings are:

’U’ DB2 folds the authorization name or password to uppercase before sending it to the data source.

’N’ DB2 does not fold the authorization name or password.

’L’ DB2 folds the authorization name or password to lowercase before sending it to the data source.

null DB2 first sends the authorization name or password as uppercase; if that fails, DB2 folds it to lowercase and sends it again.

The null setting might seem attractive because it covers many possibilities. However, from a performance perspective, it is best to set these options so that only one attempt is made for connections. If both the fold_id and fold_pw options are set to null, it is possible that DB2 will make four attempts to send the authorization name and password:

1. Both authorization name and password in uppercase.
2. Authorization name in uppercase and password in lowercase.
3. Authorization name in lowercase and password in uppercase.
4. Both authorization name and password in lowercase.

APPC Security Settings

If you are connecting to a DRDA data source, across APPC, that requires a user ID and password, or if your authentication setting is DCS and you are authenticating at a DRDA data source, ensure that your APPC security setting is PROGRAM for the connection between DB2 and that data source.

Federated Database Authentication Example

This section provides an overview of federated system authentication and authorization steps. See Figure 23 on page 304 for an overview of federated database authentication and authorization processing.

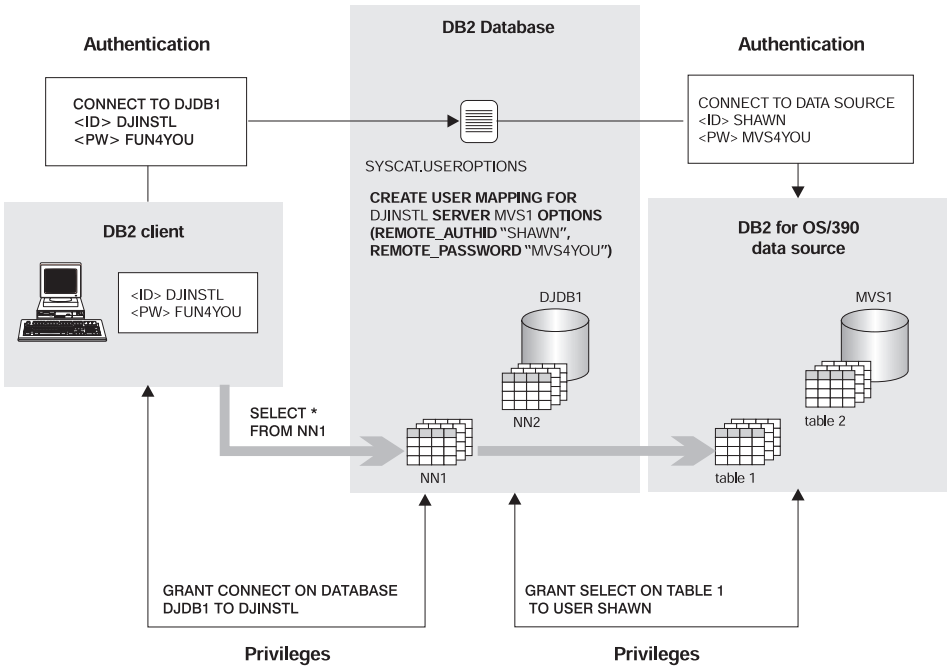


Figure 23. Federated Database Authentication and Authorization Processing

The task in this scenario is to enable the user DJINSTL to perform a UNION operation against two nicknames (NN1 and NN2). The nicknames represent two tables. One data source is a DB2 for OS/390 system where DJINSTL has a different user ID and password (see Figure 23) named MVS1. A user mapping will be required to access information at MVS1. The other data source is a DB2 system where DJINSTL's ID and password are the same. This data source, DB21, simply requires that the user ID and password are sent in uppercase.

DB2 authentication is set to SERVER. DJINSTL will access DB2 from a Windows NT client across a TCP/IP connection. The connection from DB2 to DB2 for OS/390 is also TCP/IP. The federated database name is DJDB1.

First ensure that DB2 is expecting a password and that a password is being sent. Also, ensure that the client and server authentication types match. Check the DB2 server authentication type by issuing the command:

```
GET DATABASE MANAGER CONFIGURATION
```

from the DB2 server. Check the client authentication type by issuing the command:

```
LIST DATABASE DIRECTORY
```

from the client. In both cases, ensure that authentication is set to SERVER. If the setting for the client is DCS or CLIENT, you can change it by using the UNCATALOG DATABASE and CATALOG DATABASE commands.

Next, ensure that passwords will be sent to the data sources. After connecting to the federated database DJDB1, issue the commands:

```
ALTER SERVER MVS1 OPTIONS (SET password 'Y')
ALTER SERVER DB21 OPTIONS (SET password 'Y')
```

Next, ensure that passwords are sent to the DB21 data source in the proper case:

```
ALTER SERVER DB21 OPTIONS (ADD fold_id 'U')
ALTER SERVER DB21 OPTIONS (ADD fold_pw 'U')
```

The next step is to grant privileges allowing the user DJINSTL to connect to the federated database DJDB1 and select nicknames:

```
GRANT CONNECT ON DATABASE DJDB1 TO DJINSTL;
```

Now, map DJINSTL's DB2 ID and password to the correct user ID and password for the MVS1 server:

```
CREATE USER MAPPING FOR "DJINSTL" SERVER MVS1 OPTIONS (REMOTE_AUTHID "SHAWN",
REMOTE_PASSWORD "MVS4YOU")
```

At this point, the DB2 user ID DJINSTL can send requests to data sources. Additional steps might be required to access data source objects referenced by nicknames (privileges are usually required for tables and views referenced by nicknames).

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 307
- "Database Administration Authority (DBADM)" on page 310
- "System Control Authority (SYSCTRL)" on page 308
- "System Maintenance Authority (SYSMAINT)" on page 309.

The following types of privileges exist:

- “Database Privileges” on page 310
- “Schema Privileges” on page 312
- “Table and View Privileges” on page 314
- “Nickname Privileges” on page 316
- “Server Privileges” on page 317
- “Package Privileges” on page 317
- “Index Privileges” on page 318.

Figure 24 illustrates the relationship between authorities and their span of control (database, database manager).

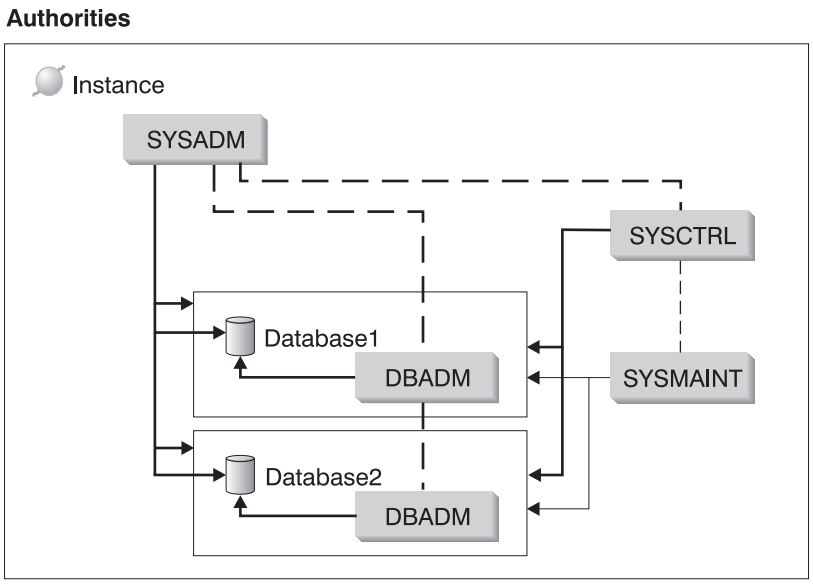


Figure 24. Hierarchy of Authorities

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 322.

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 318.) 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.

Refer to the *Command Reference*, the *Administrative 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, servers, 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 (refer to "Configuring DB2" in *Administration Guide*,

Performance). Membership in that group is controlled outside the database manager through the security facility used on your platform. Refer to 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)"
- "System Maintenance Authority (SYSMAINT)" on page 309
- "Database Administration Authority (DBADM)" on page 310

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 (refer to "Configuring DB2" in *Administration Guide, Performance*). 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 (refer to "Configuring DB2" in *Administration Guide, Performance*). 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

Figure 25 on page 311 shows the database privileges.

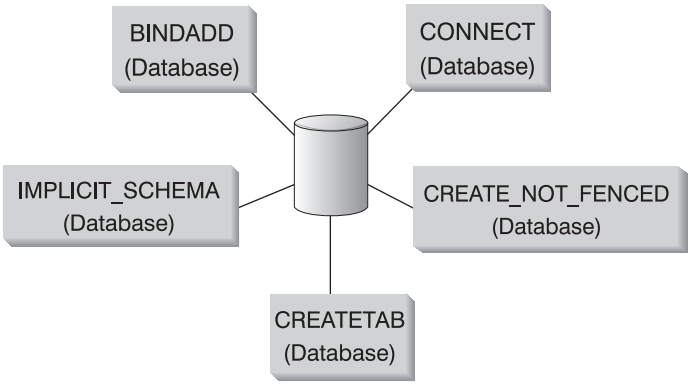


Figure 25. Database Privileges

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.) (Refer to the *Application Development 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 are in the object privilege category. Object privileges are shown in Figure 26 on page 313.

Object privileges

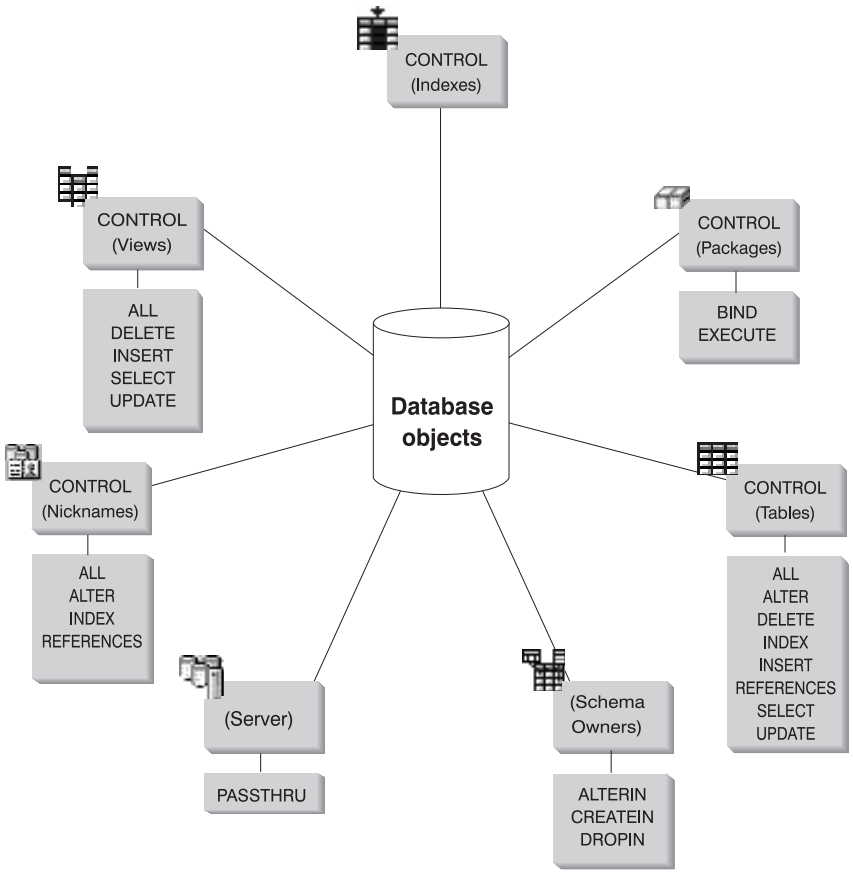


Figure 26. Object Privileges

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 318.
- 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 might 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
```

When working with typed tables, there are implications regarding table and view privileges.

Note: Privileges may be granted independently at every level of a table hierarchy. As a result, a user granted a privilege on a supertable within a hierarchy of typed tables may also indirectly affect any subtables. However, a user can only operate directly on a subtable if the necessary privilege is held on that subtable.

The supertable/subtable relationships among the tables in a table hierarchy mean that operations such as SELECT, UPDATE, and DELETE will affect the rows of the operation's target table and all its subtables (if any). This behavior can be called "substitutability". For example, suppose that you have created an Employee table of type Employee_t with a subtable Manager of type Manager_t. A manager is a (specialized) kind of employee, as indicated by the type/subtype relationship between the structured types Employee_t and Manager_t and the corresponding table/subtable relationship between the tables Employee and Manager. As a result of this relationship, the SQL query:

```
SELECT * FROM Employee
```

will return the object identifier and Employee_t attributes for both employees and managers. Similarly, the update operation:

```
UPDATE Employee SET Salary = Salary + 1000
```

will give a thousand dollar raise to managers as well as regular employees.

A user with SELECT privilege on Employee will be able to perform this SELECT operation even if they do not have an explicit SELECT privilege on Manager. However, such a user will not be permitted to perform a SELECT operation directly on the Manage subtable, and will therefore not be able to access any of the non-inherited columns of the Manager table.

Similarly, a user with UPDATE privilege on Employee will be able to perform an UPDATE operation on Employee, thereby affecting both regular employees and managers, even without having the explicit UPDATE privilege on the Manager table. However, such a user will not be permitted to perform

UPDATE operations directly on the Manager subtable, and will therefore not be able to update non-inherited columns of the Manager table.

The following manuals provide information about the authorizations required to execute specific commands, APIs, or SQL statements:

- *SQL Reference*
- *Command Reference*
- *Administrative API Reference.*

Refer to *Administration Guide, Performance* for information about the authorization required to update catalog statistics.

For information about how view privileges are determined, refer to the CREATE VIEW statement in the *SQL Reference* manual.

Nickname Privileges

Nickname privileges involve actions on nicknames in a database. These privileges do not affect privileges on the data source objects referenced by nicknames. 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 nickname including the ability to drop it, and to grant and revoke individual nickname privileges. You must have SYSADM or DBADM authority to grant CONTROL. The creator of a nickname automatically receives CONTROL privilege on the nickname.
- ALTER allows the user to change column names in a nickname, add or change the DB2 type that the column's data type maps to, and set column options for nickname columns.
- INDEX allows the user to create an index specification on a nickname. Creators of index specifications automatically have CONTROL privilege on the index.
- REFERENCES allows the user to create and drop a foreign key, specifying the nickname as the parent in a relationship. The user may have this privilege only on specific columns.

The privilege to grant these privileges to others can also be granted using the WITH GRANT OPTION on the GRANT statement.

Note: When a user or group is granted CONTROL privilege on a nickname, all other privileges on that nickname are automatically granted WITH GRANT OPTION. If you subsequently revoke the CONTROL privilege on the nickname from a user, that user will still retain the other privileges that were automatically granted.

To access data source data, you must also have the proper authorization for the objects at data sources referenced by nicknames.

When a user accesses a view that references one or more nicknames, that user must be authorized to access the view and the objects that the nicknames reference at data sources.

Server Privileges

There is one server privilege: PASSTHRU. This privilege controls which authorization IDs can issue DDL and DML statements directly (pass-through operations) to data sources.

DB2 provides two SQL statements to control pass-through operations:

- GRANT PASSTHRU, which grants the authority to issue SET PASSTHRU statements against a data source and pass-through DML and DDL statements to that data source.
- REVOKE PASSTHRU, which revokes the authority to issue SET PASSTHRU statements against a data source and pass-through DML and DDL statements to that data source.

A sample statement granting pass-through authorization to the user SHAWN for the server ORACLE1 is:

```
GRANT PASSTHRU ON SERVER ORACLE1 TO USER SHAWN
```

For complete information on the syntax of PASSTHRU statements, see the *SQL Reference*.

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.

Users with the authority to execute a package containing nicknames don't need additional privileges or an authority level for the nicknames within the package; however, they will need to pass authentication checks at the data sources containing the objects referenced by the nicknames. In addition, package users must have the appropriate privileges or authority levels for data source objects at the data source.

It is possible that packages containing nicknames might require additional authorization steps because DB2 uses dynamic SQL when communicating with DB2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source. See the *SQL Reference* for more information about how DB2 processes static and dynamic SQL.

Index Privileges

The creator of an index or an index specification 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 314).

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 328.

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"
- "Revoking Privileges" on page 320
- "Managing Implicit Authorizations by Creating and Dropping Objects" on page 321
- "Allowing Indirect Privileges through a Package" on page 322
- "Controlling Access to Data with Views" on page 324
- "Monitoring Access to Data Using the Audit Facility" on page 327.

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.

Refer to 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, refer to 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. Refer to 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, refer to the CREATE VIEW statement in the *SQL Reference* manual.

Establishing Ownership of a Plan or a Package

The BIND and PRECOMPILE commands create or change an application package. On either one, use the OWNER option to name the owner of the resulting package. There are simple rules for naming the owner of a package:

- Any user can name themselves as the owner. This is the default if the OWNER option is not specified.
- An ID with SYSADM or DBADM authority can name any authorization ID as the owner using the OWNER option.

Not all operating systems that can bind a package using DB2 database products support the OWNER option.

Refer to the *Command Reference* for more information on the BIND and PRECOMPILE commands.

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.

Allowing Indirect Privileges through a Package Containing Nicknames

When a package contains references to nicknames, authorization processing for package creators and package users is slightly more complex. When a package creator successfully binds packages that contain nicknames, the package creator does not have to pass authentication checking or privilege checking for the tables and views that the nicknames reference at the data source. However, the package executor must pass authentication and authorization checking at data sources.

For example, assume that a package creator's .SQC file contains several SQL statements. One static statement references a local table. Another dynamic statement references a nickname. When the package is bound, the package creator's authid is used to verify privileges for the local table–but no checking is done for the data source objects that the nickname identifies. When another user executes the package, assuming they have the EXECUTE privilege for that package, that user does not have to pass any additional privilege checking for the statement referencing the table. However, for the statement referencing the nickname, the user executing the package must pass authentication checking and privilege checking at the data source.

When the .SQC file contains all dynamic SQL statements and a mixture of table and nickname references, DB2 authorization checking for local objects and nicknames is similar. Package users must pass privilege checking for any local objects (tables, views) within the statements and also pass privilege checking for nickname objects (package users must pass authentication and privilege checking at the data source containing the objects that the nicknames identify). In both cases, users of the package must have the EXECUTE privilege.

The ID and password of the package executor is used for all data source authentication and privilege processing. This information can be changed by creating a user mapping.

Note: Nicknames cannot be specified in static SQL. Do not use the DYNAMICRULES option (set to BIND) with packages containing nicknames.

It is possible that packages containing nicknames might require additional authorization steps because DB2 uses dynamic SQL when communicating

with DB2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source. See the *SQL Reference* for more information about how DB2 processes static and dynamic SQL.

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.
- Access only to a subset of the rows or columns in data source tables or views. If you are accessing data sources through nicknames, you can create local DB2 views that reference nicknames. These views can reference nicknames from one or many data sources.

  Note: Because you can create a view that contains nickname references for more than one data source, your users can access data in multiple data sources from one view. These views are called *multi-location views*. Such views are useful when joining information in columns of sensitive tables across a distributed environment or when individual users lack the privileges needed at data sources for specific objects.

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 182 for more information.

If you are creating views that reference nicknames, you do not need additional authority on the data source objects (tables and views) referenced by nicknames in the view; however, your users must have SELECT authority or the equivalent authorization level for the underlying data source objects when they access the view.

If your users do not have the proper authority at the data source for underlying objects (tables and views), you can:

- Create a data source view over those columns in the data source table that are OK for the user to access

- Grant the SELECT privilege on this view to users
- Create a nickname to reference the view

Users can then access the columns by issuing a SELECT statement that references the new nickname.

The following scenario provides a more detailed example of how views can be used to restrict access to information.

Many people might 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

NAME	LOCATION
Quill	Denver
Davis	Denver
Edwards	Denver
Gafney	Denver

Monitoring Access to Data Using the Audit Facility

The DB2 audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. While not a facility that prevents access to data, the audit facility can monitor and keep a record of attempts to access or modify data objects.

SYSADM authority is required to use the audit facility administrator tool, `db2audit`.

See "Chapter 7. Auditing DB2 Activities" on page 333 for a detailed description of the DB2 audit facility.

Tasks and Required Authorizations

Not all organizations divide job responsibilities in the same manner. Table 25 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 25. 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.

Table 25. Common Job Titles, Tasks, and Required Authorization (continued)

JOB TITLE	TASKS	REQUIRED AUTHORIZATION
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
SYSCAT.PASSTHRUAUTH	Lists the server privilege

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 330
- "Retrieving Names Authorized to Access a Table" on page 330
- "Retrieving All Privileges Granted to Users" on page 330
- "Securing the System Catalog Views" on page 331.

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
UNION
SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SERVER  ' FROM SYSCAT.PASSTHRUAUTH
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. Refer to 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 310 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 following catalog views:

- SYSCAT.DBAUTH
- SYSCAT.TABAUTH
- SYSCAT.PACKAGEAUTH
- SYSCAT.INDEXAUTH
- SYSCAT.COLAUTH
- SYSCAT.PASSTHRUAUTH
- SYSCAT.SCHEMAAUTH

This would prevent information on user privileges, which could be used to target an authorization name for break-in, from becoming available to everyone with access to the database.

You should also examine the columns for which statistics are gathered (refer to “Catalog Statistics” in the *Administration Guide, Performance*). 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 7. Auditing DB2 Activities

Authentication, authorities, and privileges can be used to control known or anticipated access to data, but these methods may be insufficient to prevent unknown or unanticipated access to data. To assist in the detection of this latter type of data access, DB2 provides an audit facility. Successful monitoring of unwanted data access and subsequent analysis can lead to improvements in the control of data access and the ultimate prevention of malicious or careless unauthorized access to the data. The monitoring of application and individual user access, including system administration actions, can provide a historical record of activity on your database systems.

The DB2 audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. The analysis of these records can reveal usage patterns which would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse.

The audit facility acts at an instance level, recording all instance level activities and database level activities.

When working in a partitioned database environment, many of the auditable events occur at the partition at which the user is connected (the coordinator node) or at the catalog node (if they are not the same partition). The implication of this is that audit records can be generated by more than one partition. Part of each audit record contains information on the coordinator node and originating node identifiers.

The audit log (db2audit.log) and the audit configuration file (db2audit.cfg) are located in the instance's `security` subdirectory. At the time you create an instance, read/write permissions are set on these files, where possible, by the operating system. By default, the permissions are read/write for the instance owner only. It is recommended that you do not change these permissions.

Users of the audit facility administrator tool, db2audit, must have SYSADM authority/privileges.

The audit facility must be stopped and started explicitly. When starting, the audit facility uses existing audit configuration information. Since the audit facility is independent of the DB2 server, it will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log.

Authorized users of the audit facility can control the following actions within the audit facility:

- Start recording auditable events within the DB2 instance.
- Stop recording auditable events within the DB2 instance.
- Configure the behavior of the audit facility, including selecting the categories of the auditable events to be recorded.
- Request a description of the current audit configuration.
- Flush any pending audit records from the instance and write them to the audit log.
- Extract audit records by formatting and copying them from the audit log to a flat file or ASCII delimited files. Extraction is done for one of two reasons: In preparation for analysis of log records or in preparation for pruning of log records.
- Prune audit records from the current audit log.

There are different categories of audit records that may be generated. In the description of the categories of events available for auditing (below), you should notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:

- Audit (AUDIT). Generates records when audit settings are changed or when the audit log is accessed.
- Authorization Checking (CHECKING). Generates records during authorization checking of attempts to access or manipulate DB2 objects or functions.
- Object Maintenance (OBJMAINT). Generates records when creating or dropping data objects.
- Security Maintenance (SECMAINT). Generates records when granting or revoking: Object or database privileges, or DBADM authority. Records are also generated when the database manager security configuration parameters SYSADM_GROUP, SYSCTRL_GROUP, or SYSMAINT_GROUP are modified.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.
- User Validation (VALIDATE). Generates records when authenticating users or retrieving system security information.
- Operation Context (CONTEXT). Generates records to show the operation context when a database operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation. For example, an SQL statement for dynamic SQL, a

package identifier for static SQL, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results.

Note: The SQL statement providing the operation context might be very long and is completely shown within the CONTEXT record. This can make the CONTEXT record very large.

- You can audit failures, successes, or both.

Any operation on the database may generate several records. The actual number of records generated and moved to the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.

Audit Facility Behavior

The audit facility records auditable events including those affecting database instances. For this reason, the audit facility is an independent part of DB2 that can operate even if the DB2 instance is stopped. If the audit facility is active, then when a stopped instance is started, auditing of database events in the instance resumes.

The timing of the writing of audit records to the audit log can have a significant impact on the performance of databases in the instance. The writing of the audit records can take place synchronously or asynchronously with the occurrence of the events causing the generation of those records. The value of the AUDIT_BUF_SZ database manager configuration parameter determines when the writing of audit records is done.

If the value of this parameter is zero (0), the writing is done synchronously. The event generating the audit record will wait until the record is written to disk. The wait associated with each record causes the performance of DB2 to decrease.

If the value of AUDIT_BUF_SZ is greater than zero, the record writing is done asynchronously. The value of the AUDIT_BUF_SZ when it is greater than zero is the number of 4 KB pages used to create an internal buffer. The internal buffer is used to keep a number of audit records before writing a group of them out to disk. The statement generating the audit record as a result of an audit event will not wait until the record is written to disk, and can continue its operation.

In the asynchronous case, it could be possible for audit records to remain in an unfilled buffer for some time. To prevent this from happening for an

extended period, the database manager will force the writing of the audit records regularly. An authorized user of the audit facility may also flush the audit buffer with an explicit request.

There are differences when an error occurs dependent on whether there is synchronous or asynchronous record writing. In asynchronous mode there may be some records lost because the audit records are buffered before being written to disk. In synchronous mode there may be one record lost because the error could only prevent at most one audit record from being written.

The setting of the ERRORTYPE audit facility parameter controls how errors are managed between DB2 and the audit facility. When the audit facility is active, and the setting of the ERRORTYPE audit facility parameter is `AUDIT`, then the audit facility is treated in the same way as any other part of DB2. An audit record must be written (to disk in synchronous mode; or to the audit buffer in asynchronous mode) for an audit event associated with a statement to be considered successful. Whenever an error is encountered when running in this mode, a negative SQLCODE is returned to the application for the statement generating an audit record. If the error type is set to `NORMAL`, then any error from db2audit is ignored and the operation's SQLCODE is returned. See "Audit Facility Usage Scenarios" on page 337 for additional details on the ERRORTYPE audit facility parameters (and other related parameters).

Depending on the API or SQL statement and the audit settings for the DB2 instance, none, one, or several audit records may be generated for a particular event. For example, an SQL UPDATE statement with a SELECT subquery may result in one audit record containing the results of the authorization check for UPDATE privilege on a table and another record containing the results of the authorization check for SELECT privilege on a table.

For dynamic data manipulation language (DML) statements, audit records are generated for all authorization checking at the time that the statement is prepared. Reuse of those statements by the same user will not be audited again since no authorization checking takes place at that time. However, if a change has been made to one of the catalog tables containing privilege information, then in the next unit of work, the statement privileges for the cached dynamic SQL statements are re-checked and one or more new audit records created.

For a package containing only static DML statements, the only auditable event that could generate an audit record is the authorization check to see if a user has the privilege to execute that package. The authorization checking and possible audit record creation required for the static SQL statements in the package is carried out at the time the package is precompiled or bound. The execution of the static SQL statements within the package is not auditable.

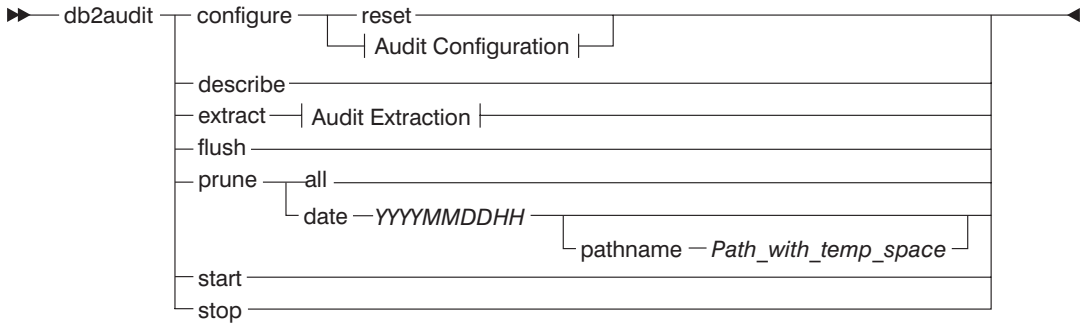
When a package is re-bound either explicitly by the user, or implicitly by the system, audit records are generated for the authorization checks required by the static SQL statements.

For statements where authorization checking is performed at statement execution time (for example, data definition language (DDL), GRANT, and REVOKE statements), audit records are generated whenever these statements are used.

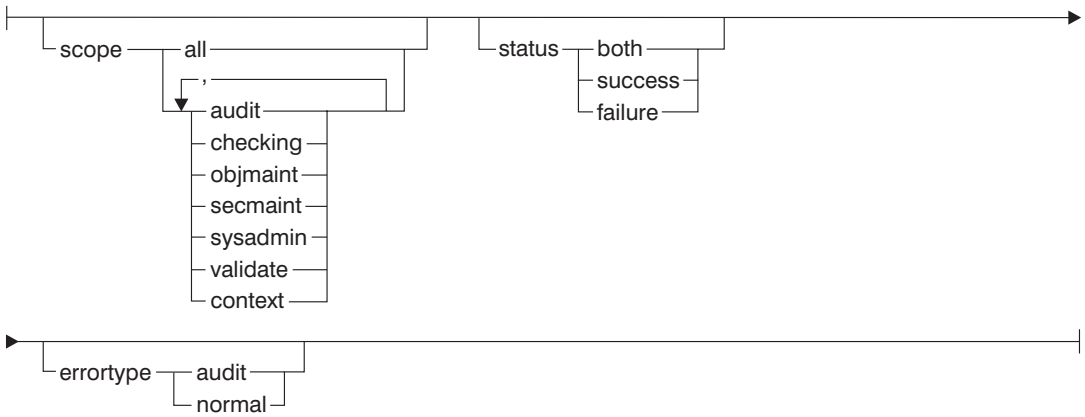
Note: When executing DDL, the section number recorded for all events (except the context events) in the audit record will be zero (0) no matter what the actual section number of the statement might have been.

Audit Facility Usage Scenarios

By considering the syntax of the audit facility, we can review the way the facility can be used.



Audit Configuration:



Audit Extraction:

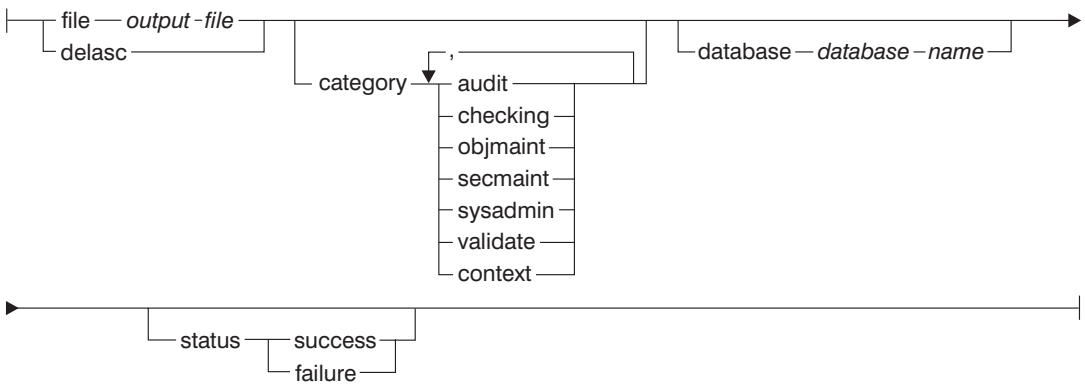


Figure 27. DB2AUDIT Syntax

The following is a description and the implied use of each parameter:

configure

This parameter allows the modification of the db2audit.cfg

configuration file in the instance's *security* subdirectory. Updates to this file can occur even when the instance is shut down. Updates occurring when the instance is active dynamically affect the auditing being done by DB2 across all partitions. The configure action on the configuration file causes the creation of an audit record if the audit facility has been started and the *audit* category of auditable events is being audited.

The following are the possible actions on the configuration file:

- RESET. This action causes the configuration file to revert to the initial configuration (where SCOPE is all of the categories except CONTEXT, STATUS is FAILURE, ERRORTYPE is NORMAL, and AUDIT is OFF). This action will create a new audit configuration file if the original has been lost or damaged.
- SCOPE. This action specifies which category or categories of events are to be audited. This action also allows a particular focus for auditing and reduces the growth of the log. It is recommended that the number and type of events being logged be limited as much as possible, otherwise the audit log will grow rapidly.

  Note: Please notice that the default SCOPE is all categories except CONTEXT and may result in records being generated rapidly. In conjunction with the mode (synchronous or asynchronous), the selection of the categories may result in a significant performance reduction and significantly increased disk requirements.
- STATUS. This action specifies whether only successful or failing events, or both successful and failing events, should be logged.

  Note: Context events occur before the status of an operation is known. Therefore, such events are logged regardless of the value associated with this parameter.
- ERRORTYPE. This action specifies whether audit errors are returned to the user or are ignored. The value for this parameter can be:
  - AUDIT. All errors including errors occurring within the audit facility are managed by DB2 and all negative SQLCODEs are reported back to the caller.
  - NORMAL. Any errors generated by db2audit are ignored and only the SQLCODEs for the errors associated with the operation being performed are returned to the application.

describe

This parameter displays to standard output the current audit configuration information and status.

extract This parameter allows the movement of audit records from the audit log to an indicated destination. If no optional clauses are specified, then all of the audit records are extracted and placed in a flat report file. If the "extract" parameter is not specified, the audit record is placed a file called `db2audit.out` in the `security` directory. If `output_file` already exists, an error message is returned.

The following are the possible options that can be used when extracting:

- FILE. The extracted audit records are placed in a file (output_file).
- DELASC. The extracted audit records are placed in a delimited ASCII format suitable for loading into DB2 relational tables. The output is placed in separate files: one for each category. The filenames are:
  - audit.del
  - checking.del
  - objmaint.del
  - secmaint.del
  - sysadmin.del
  - validate.del
  - context.del

  The DELASC choice also allows you to override the default audit character string delimiter ("0xff") when extracting from the audit log. You would use DELASC DELIMITER followed by the new delimiter that you wish to use in preparation for loading into a table that will hold the audit records. The new load delimiter can be either a single character (such as !) or a four-byte string representing a hexadecimal number (such as 0xff). For more information, refer to "Audit Facility Tips and Techniques" on page 356.
- CATEGORY. The audit records for the specified categories of audit events are to be extracted. If not specified, all categories are eligible for extraction.
- DATABASE. The audit records for a specified database are to be extracted. If not specified, all databases are eligible for extraction.
- STATUS. The audit records for the specified status are to be extracted. If not specified, all records are eligible for extraction.

flush This parameter forces any pending audit records to be written to the audit log. Also, the audit state is reset in the engine from "unable to log" to a state of "ready to log" if the audit facility is in an error state.

prune This parameter allows for the deletion of audit records from the audit

log. If the audit facility is active and the “audit” category of events has been specified for auditing, then an audit record will be logged after the audit log is pruned.

The following are the possible options that can be used when pruning:

- ALL. All of the audit records in the audit log are to be deleted.
- DATE yyyymmddhh. The user can specify that all audit records that occurred on or before the date/time specified are to be deleted from the audit log. The user may optionally supply a

  `pathname`

  which the audit facility will use as a temporary space when pruning the audit log. This temporary space allows for the pruning of the audit log when the disk it resides on is full and does not have enough space to allow for a pruning operation.

start This parameter causes the audit facility to begin auditing events based on the contents of the db2audit.cfg file. In a partitioned DB2 instance, auditing will begin on all partitions when this clause is specified. If the “audit” category of events has been specified for auditing, then an audit record will be logged when the audit facility is started.

stop This parameter causes the audit facility to stop auditing events. In a partitioned DB2 instance, auditing will be stopped on all partitions when this clause is specified. If the “audit” category of events has been specified for auditing, then an audit record will be logged when the audit facility is stopped.

Audit Facility Messages

SQL1322N An error occurred when writing to the audit log file.

Explanation: The DB2 audit facility encountered an error when invoked to record an audit event to the audit log file. There is no space on the file system where the audit log resides.

User Response: The system administrator should free up space on this file system or prune the audit log to reduce its size.

When more space is available, use db2audit to flush out any data in memory, and to reset the auditor to a ready state. Ensure that appropriate extracts have occurred, or a copy of the log has been made before pruning the log, as deleted records are not recoverable.

sqlcode: 1322

sqlstate: 50030

SQL1323N An error occurred when accessing the audit configuration file.

Explanation: The audit configuration file (db2audit.cfg) could not be opened, or was invalid. Possible reasons for this error are that the db2audit.cfg file either does not exist, or has been damaged.

User Response: Take one of the following actions:

- Restore from a saved version of the file.
- Reset the audit facility configuration file by issuing

  ```
  db2audit reset
  ```

sqlcode: 1323

sqlstate: 57019

Audit Facility Record Layouts

When an audit record is extracted from the audit log using the DELASC extract option, each record will have one of the formats shown in the following tables. Each table will begin by showing the contents of a sample record. The description of each item of the record is shown one row at a time in the associated table. If the item is important, the name of the item will be highlighted (**bold**). These items contain information that are of most interest to you.

Notes:

1. Not all fields in the sample records will have values.
2. Some fields such as "Access Attempted" are stored in the delimited ASCII format as bitmaps. In this flat report file, however, these fields will appear as a set of strings representing the bitmap values.

Table 26. Audit Record Layout for AUDIT Events

```
timestamp=1998-06-24-11.54.05.151232;category=AUDIT;audit event=START;
  event correlator=0;event status=0;
  userid=boss;authid=BOSS;
```

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: AUDIT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: CONFIGURE, DB2AUD, EXTRACT, FLUSH, PRUNE, START, STOP, and UPDATE_ADMIN_CFG
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.

Table 27. Audit Record Layout for CHECKING Events

```
timestamp=1998-06-24-08.42.11.622984;category=CHECKING;audit event=CHECKING_OBJECT;
  event correlator=2;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  object name=FOO;object type=DATABASE;
  access approval reason=DATABASE;access attempted=CONNECT;
```

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: CHECKING
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: CHECKING_OBJECT and CHECKING_FUNCTION
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application Name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR (128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR (128)	Name of object for which the audit event was generated.

Table 27. Audit Record Layout for CHECKING Events (continued)

```
timestamp=1998-06-24-08.42.11.622984;category=CHECKING;audit event=CHECKING_OBJECT;
  event correlator=2;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  object name=FOO;object type=DATABASE;
  access approval reason=DATABASE;access attempted=CONNECT;
```

NAME	FORMAT	DESCRIPTION
Object Type	VARCHAR (32)	Type of object for which the audit event was generated. Possible values include: TABLE, VIEW, ALIAS, FUNCTION, INDEX, PACKAGE, DATA_TYPE, NODEGROUP, SCHEMA, STORED_PROCEDURE, BUFFERPOOL, TABLESPACE, EVENT_MONITOR, TRIGGER, DATABASE, INSTANCE, FOREIGN_KEY, PRIMARY_KEY, UNIQUE_CONSTRAINT, CHECK_CONSTRAINT, WRAPPER, SERVER, NICKNAME, USER MAPPING, SERVER OPTION, TYPE MAPPING, FUNCTION MAPPING, SUMMARY TABLES, and NONE.
Access Approval Reason	CHAR(18)	Indicates the reason why access was approved for this audit event. Possible values include: Those shown in the first list following this table.
Access Attempted	CHAR(18)	Indicates the type of access that was attempted. Possible values include: Those shown in the second list following this table.

The following is the list of possible CHECKING access approval reasons:

0x0000000000000001 ACCESS DENIED
Access is not approved; rather, it was denied.

0x0000000000000002 SYSADM
Access is approved; the application / user has SYSADM authority.

0x0000000000000004 SYSCTRL
Access is approved; the application / user has SYSCTRL authority.

0x0000000000000008 SYSMAINT
Access is approved; the application / user has SYSMAINT authority.

0x0000000000000010 DBADM
Access is approved; the application / user has DBADM authority.

0x0000000000000020 DATABASE PRIVILEGE
Access is approved; the application / user has an explicit privilege on the database.

0x0000000000000040 OBJECT PRIVILEGE
Access is approved; the application / user has an explicit privilege on the object or function.

0x0000000000000080 DEFINER
Access is approved; the application/user is the definer of the object or function.

0x0000000000000100 OWNER
Access is approved; the application/user is the owner of the object or function.

0x0000000000000200 CONTROL
Access is approved; the application/user has CONTROL privilege on the object or function.

0x0000000000000400 BIND
Access is approved; the application/user has bind privilege on the package.

The following is the list of possible CHECKING access attempted types:

0x0000000000000002 ALTER
Attempt to alter an object.

0x0000000000000004 DELETE
Attempt to delete an object.

0x0000000000000008 INDEX
Attempt to use an index.

0x0000000000000010 INSERT
Attempt to insert into an object.

0x0000000000000020 SELECT
Attempt to query a table or view.

0x0000000000000040 UPDATE
Attempt to update data in an object.

0x0000000000000080 REFERENCE
Attempt to establish referential constraints between objects.

0x0000000000000100 CREATE
Attempt to create an object.

0x0000000000000200 DROP
Attempt to drop an object.

0x0000000000000400 CREATEIN
Attempt to create an object within another schema.

0x0000000000000800 DROPIN
Attempt to drop an object found within another schema.

0x0000000000001000 ALTERIN
Attempt to alter or modify an object found within another schema.

0x0000000000002000 EXECUTE
Attempt to execute or run an application.

0x0000000000004000 BIND
Attempt to bind or prepare an application.

0x0000000000008000 SET EVENT MONITOR
Attempt to set event monitor switches.

0x0000000000010000 SET CONSTRAINTS
Attempt to set constraints on an object.

0x0000000000020000 COMMENT ON
Attempt to create comments on an object.

0x0000000000040000 GRANT
Attempt to grant privileges on an object to another user ID.

0x0000000000080000 REVOKE
Attempt to revoke privileges on an object from a user ID.

0x0000000000100000 LOCK
Attempt to lock an object.

0x0000000000200000 RENAME
Attempt to rename an object.

0x0000000000400000 CONNECT
Attempt to connect to an object.

0x0000000000800000 Member of SYS Group
Attempt to access or use a member of the SYS group.

0x0000000001000000 USE
Attempt to create a table in a table space.

Table 28. Audit Record Layout for OBJMAINT Events

```
timestamp=1998-06-24-08.42.41.957524;category=OBJMAINT;audit event=CREATE_OBJECT;
  event correlator=3;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  package schema=NULLID;package name=SQLC28A1;
  package section=0;object schema=BOSS;object name=AUDIT;object type=TABLE;
```

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: OBJMAINT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: CREATE_OBJECT and DROP_OBJECT

Table 28. Audit Record Layout for OBJMAINT Events (continued)

```
timestamp=1998-06-24-08.42.41.957524;category=OBJMAINT;audit event=CREATE_OBJECT;
  event correlator=3;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  package schema=NULLID;package name=SQLC28A1;
  package section=0;object schema=BOSS;object name=AUDIT;object type=TABLE;
```

NAME	FORMAT	DESCRIPTION
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application Name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR (128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR (128)	Name of object for which the audit event was generated.
Object Type	VARCHAR (32)	Type of object for which the audit event was generated. Possible values include: TABLE, VIEW, ALIAS, FUNCTION, INDEX, PACKAGE, DATA_TYPE, NODEGROUP, SCHEMA, STORED_PROCEDURE, BUFFERPOOL, TABLESPACE, EVENT_MONITOR, TRIGGER, DATABASE, INSTANCE, FOREIGN_KEY, PRIMARY_KEY, UNIQUE_CONSTRAINT, CHECK_CONSTRAINT, WRAPPER, SERVER, NICKNAME, USER MAPPING, SERVER OPTION, TYPE MAPPING, FUNCTION MAPPING, SUMMARY TABLES, and NONE.

Table 29. Audit Record Layout for SECMAINT Events

```
timestamp=1998-06-24-11.57.45.188101;category=SECMAINT;audit event=GRANT;
  event correlator=4;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.boss.980624155728;application name=db2bp;
  package schema=NULLID;package name=SQLC28A1;
  package section=0;object schema=BOSS;object name=T1;object type=TABLE;
  grantor=BOSS;grantee=WORKER;grantee type=USER;privilege=SELECT;
```

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: SECMAINT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: GRANT, REVOKE, IMPLICIT_GRANT, IMPLICIT_REVOKE, and UPDATE_DBM_CFG.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application Name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR (128)	Schema of the object for which the audit event was generated.

Table 29. Audit Record Layout for SECMAINT Events (continued)

```
timestamp=1998-06-24-11.57.45.188101;category=SECMAINT;audit event=GRANT;
  event correlator=4;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.boss.980624155728;application name=db2bp;
  package schema=NULLID;package name=SQLC28A1;
  package section=0;object schema=BOSS;object name=T1;object type=TABLE;
  grantor=BOSS;grantee=WORKER;grantee type=USER;privilege=SELECT;
```

NAME	FORMAT	DESCRIPTION
Object Name	VARCHAR (128)	Name of object for which the audit event was generated.
Object Type	VARCHAR (32)	Type of object for which the audit event was generated. Possible values include: TABLE, VIEW, ALIAS, FUNCTION, INDEX, PACKAGE, DATA_TYPE, NODEGROUP, SCHEMA, STORED_PROCEDURE, BUFFERPOOL, TABLESPACE, EVENT_MONITOR, TRIGGER, DATABASE, INSTANCE, FOREIGN_KEY, PRIMARY_KEY, UNIQUE_CONSTRAINT, CHECK_CONSTRAINT, WRAPPER, SERVER, NICKNAME, USER MAPPING, SERVER OPTION, TYPE MAPPING, FUNCTION MAPPING, SUMMARY TABLES, and NONE.
Grantor	VARCHAR (128)	Grantor ID.
Grantee	VARCHAR (128)	Grantee ID for which a privilege or authority was granted or revoked.
Grantee Type	VARCHAR (32)	Type of the grantee that was granted to or revoked from. Possible values include: USER, GROUP, or BOTH.
Privilege or Authority	CHAR(18)	Indicates the type of privilege or authority granted or revoked. Possible values include: Those shown in the list following this table.

The following is the list of possible SECMAINT privileges or authorities:

0x0000000000000001 Control Table
Control privilege granted or revoked on a table.

0x0000000000000002 ALTER TABLE
Privilege granted or revoked to alter a table.

0x0000000000000004 ALTER TABLE with GRANT
Privilege granted or revoked to alter a table with granting of privileges allowed.

0x0000000000000008 DELETE TABLE
Privilege granted or revoked to drop a table.

0x0000000000000010 DELETE TABLE with GRANT
Privilege granted or revoked to drop a table with granting of privileges allowed.

0x0000000000000020 Table Index
Privilege granted or revoked on an index.

0x0000000000000040 Table Index with GRANT
Privilege granted or revoked on an index with granting of privileges allowed.

0x0000000000000080 Table INSERT
Privilege granted or revoked on an insert on a table.

0x0000000000000100 Table INSERT with GRANT
Privilege granted or revoked on an insert on a table with granting of privileges allowed.

0x0000000000000200 Table SELECT
Privilege granted or revoked on a select on a table.

0x0000000000000400 Table SELECT with GRANT
Privilege granted or revoked on a select on a table with granting of privileges allowed.

0x0000000000000800 Table UPDATE
Privilege granted or revoked on an update on a table.

0x0000000000001000 Table UPDATE with GRANT
Privilege granted or revoked on an update on a table with granting of privileges allowed.

0x0000000000002000 Table REFERENCE
Privilege granted or revoked on a reference on a table.

0x0000000000004000 Table REFERENCE with GRANT
Privilege granted or revoked on a reference on a table with granting of privileges allowed.

0x0000000000008000 Package BIND
Bind privilege granted or revoked on a package.

0x0000000000010000 Package EXECUTE
Execute privilege granted or revoked on a package.

0x0000000000020000 CREATEIN Schema
Createin privilege granted or revoked on a schema.

0x0000000000040000 CREATEIN Schema with GRANT
Createin privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000080000 DROPIN Schema
Dropin privilege granted or revoked on a schema.

0x0000000000100000 DROPIN Schema with GRANT
Dropin privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000200000 ALTERIN Schema
Alterin privilege granted or revoked on a schema.

0x0000000000400000 ALTERIN Schema with GRANT
Alterin privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000800000 DBADM Authority
DBADM authority granted or revoked.

0x0000000001000000 CREATETAB Authority
Createtab authority granted or revoked.

0x0000000002000000 BINDADD Authority
Bindadd authority granted or revoked.

0x0000000004000000 CONNECT Authority
Connect authority granted or revoked.

0x0000000008000000 Create not fenced Authority
Create not fenced authority granted or revoked.

0x0000000010000000 Implicit Schema Authority
Implicit schema authority granted or revoked.

0x0000000020000000 Server PASSTHRU
Privilege granted or revoked to use the pass-through facility with this server (federated database data source).

0x0000000100000000 Table Space USE
Privilege granted or revoked to create a table in a table space.

0x0000000400000000 Column UPDATE
Privilege granted or revoked on an update on one or more specific columns of a table.

0x0000000800000000 Column UPDATE with GRANT
Privilege granted or revoked on an update on one or more specific columns of a table with granting of privileges allowed.

0x0000001000000000 Column REFERENCE
Privilege granted or revoked on a reference on one or more specific columns of a table.

0x0000002000000000 Column REFERENCE with GRANT
Privilege granted or revoked on a reference on one or more specific columns of a table with granting of privileges allowed.

Table 30. Audit Record Layout for SYSADMIN Events

timestamp=1998-06-24-11.54.04.129923;**category=SYSADMIN;audit event=DB2AUDIT;**
event correlator=1;event status=0;
userid=boss;authid=BOSS;
application id=*LOCAL.boss.980624155404;application name=db2audit;		

NAME	**FORMAT**	**DESCRIPTION**
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: SYSADMIN
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: Those shown in the list following this table.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application Name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.

The following is the list of possible SYSADMIN audit events:

Table 31. SYSADMIN Audit Events

START_DB2
STOP_DB2
CREATE_DATABASE
DROP_DATABASE
UPDATE_DBM_CFG
UPDATE_DB_CFG
CREATE_TABLESPACE
DROP_TABLESPACE
ALTER_TABLESPACE
CREATE_NODEGROUP
DROP_NODEGROUP
ALTER_NODEGROUP
CREATE_BUFFERPOOL
DROP_BUFFERPOOL
ALTER_BUFFERPOOL
CREATE_EVENT_MONITOR
DROP_EVENT_MONITOR
ENABLE_MULTIPAGE
MIGRATE_DB_DIR
DB2TRC
DB2SET
ACTIVATE_DB
ADD_NODE
BACKUP_DB
CATALOG_NODE
CATALOG_DB
CATALOG_DCS_DB
CHANGE_DB_COMMENT
DEACTIVATE_DB
DROP_NODE_VERIFY
FORCE_APPLICATION
GET_SNAPSHOT
LIST_DRDA_INDOUBT_TRANSACTIONS
MIGRATE_DB
RESET_ADMIN_CFG
RESET_DB_CFG
RESET_DBM_CFG
RESET_MONITOR
RESTORE_DB
ROLLFORWARD_DB
SET_RUNTIME_DEGREE
SET_TABLESPACE_CONTAINERS
UNCATALOG_DB
UNCATALOG_DCS_DB
UNCATALOG_NODE
UPDATE_ADMIN_CFG
UPDATE_MON_SWITCHES
LOAD_TABLE
DB2AUDIT
SET_APPL_PRIORITY
CREATE_DB_AT_NODE
KILLDBM
MIGRATE_SYSTEM_DIRECTORY
DB2REMOT
DB2AUD
MERGE_DBM_CONFIG_FILE
UPDATE_CLI_CONFIGURATION
OPEN_TABLESPACE_QUERY
SINGLE_TABLESPACE_QUERY
CLOSE_TABLESPACE_QUERY
FETCH_TABLESPACE
OPEN_CONTAINER_QUERY
FETCH_CONTAINER_QUERY
CLOSE_CONTAINER_QUERY
GET_TABLESPACE_STATISTICS
DESCRIBE_DATABASE
ESTIMATE_SNAPSHOT_SIZE
READ_ASYNC_LOG_RECORD
PRUNE_RECOVERY_HISTORY
UPDATE_RECOVERY_HISTORY
QUIESCE_TABLESPACE
UNLOAD_TABLE
UPDATE_DATABASE_VERSION
CREATE_INSTANCE
DELETE_INSTANCE
SET_EVENT_MONITOR
GRANT_DBADM
REVOKE_DBADM
GRANT_DB_AUTHORITIES
REVOKE_DB_AUTHORITIES
REDIST_NODEGROUP

Table 32. Audit Record Layout for VALIDATE Events

```
timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;audit event=CHECK_GROUP_MEMBERSHIP;
  event correlator=2;event status=-1092;
  database=FOO;userid=boss;authid=BOSS;execution id=newton;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  auth type=SERVER;
```

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: VALIDATE
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: GET_GROUPS, GET_USERID, AUTHENTICATE_PASSWORD, and VALIDATE_USER.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Execution ID	VARCHAR(1024)	Execution ID in use at the time of the audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application Name in use at the time the audit event occurred.
Authentication Type	VARCHAR (32)	Authentication type at the time of the audit event.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.

Table 33. Audit Record Layout for CONTEXT Events

```
timestamp=1998-06-24-08.42.41.476840;category=CONTEXT;audit event=EXECUTE_IMMEDIATE;
  event correlator=3;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  package schema=NULLID;package name=SQLC28A1;
  package section=203;text=create table audit(c1 char(10), c2 integer);
```

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: CONTEXT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: Those shown in the list following this table.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application Name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Statement Text (statement)	CLOB (32K)	Text of the SQL statement, if applicable. Null if no SQL statement text is available.

The following is the list of possible CONTEXT audit events:

Table 34. CONTEXT Audit Events

CONNECT	SET_APPL_PRIORITY
CONNECT_RESET	RESET_DB_CFG
ATTACH	GET_DB_CFG
DETACH	GET_DFLT_CFG
DARI_START	UPDATE_DBM_CFG
DARI_STOP	SET_MONITOR
BACKUP_DB	GET_SNAPSHOT
RESTORE_DB	ESTIMATE_SNAPSHOT_SIZE
ROLLFORWARD_DB	RESET_MONITOR
OPEN_TABLESPACE_QUERY	OPEN_HISTORY_FILE
FETCH_TABLESPACE	CLOSE_HISTORY_FILE
CLOSE_TABLESPACE_QUERY	FETCH_HISTORY_FILE
OPEN_CONTAINER_QUERY	SET_RUNTIME_DEGREE
CLOSE_CONTAINER_QUERY	UPDATE_AUDIT
FETCH_CONTAINER_QUERY	DBM_CFG_OPERATION
SET_TABLESPACE_CONTAINERS	DISCOVER
GET_TABLESPACE_STATISTIC	OPEN_CURSOR
READ_ASYNC_LOG_RECORD	CLOSE_CURSOR
QUIESCE_TABLESPACE	FETCH_CURSOR
LOAD_TABLE	EXECUTE
UNLOAD_TABLE	EXECUTE_IMMEDIATE
UPDATE_RECOVERY_HISTORY	PREPARE
PRUNE_RECOVERY_HISTORY	DESCRIBE
SINGLE_TABLESPACE_QUERY	BIND
LOAD_MSG_FILE	REBIND
UNQUIESCE_TABLESPACE	RUNSTATS
ENABLE_MULTIPAGE	REORG
DESCRIBE_DATABASE	REDISTRIBUTE
DROP_DATABASE	COMMIT
CREATE_DATABASE	ROLLBACK
ADD_NODE	REQUEST_ROLLBACK
FORCE_APPLICATION	IMPLICIT_REBIND

Audit Facility Tips and Techniques

In most cases, when working with CHECKING events, the object type field in the audit record is the object being checked to see if the required privilege or authority is held by the user ID attempting to access the object. For example, if a user attempts to ALTER a table by adding a column, then the CHECKING event audit record will indicate the access attempted was “ALTER” and the object type being checked was “TABLE” (note: not the column since it is table privileges that must be checked).

However, when the checking involves verifying if a database authority exists to allow a user ID to CREATE or BIND an object, or to delete an object, then

although there is a check against the database, the object type field will specify the object being created, bound, or dropped (rather than the database itself).

When creating an index on a table, the privilege to create an index is required, therefore the CHECKING event audit record will have an access attempt type of “index” rather than “create”.

When binding a package that already exists, then an OBJMAINT event audit record is created for the DROP of the package and then another OBJMAINT event audit record is created for the CREATE of the new copy of the package.

SQL Data Definition Language (DDL) may generate OBJMAINT or SECMAINT events that are logged as successful. It is possible however that following the logging of the event, a subsequent error may cause a ROLLBACK to occur. This would leave the object as not created; or the GRANT or REVOKE actions as incomplete. The use of CONTEXT events becomes important in this case. Such CONTEXT event audit records, especially the statement that ends the event, will indicate the nature of the completion of the attempted operation.

When extracting audit records in a delimited ASCII format suitable for loading into a DB2 relational table, you should be clear regarding the delimiter used within the statement text field. This can be done when extracting the delimited ASCII file and is done using:

```
db2audit extract delasc delimiter <load delimiter>
```

The *load delimiter* can be a single character (such as ") or a four-byte string representing a hexadecimal value (such as “0xff”). Examples of valid commands are:

```
db2audit extract delasc
db2audit extract delasc delimiter !
db2audit extract delasc delimiter 0xff
```

If you have used anything other than the default load delimiter (“"”) as the delimiter when extracting, you should use the MODIFIED BY option on the LOAD command. A partial example of the LOAD command with “0xff” used as the delimiter follows:

```
db2 load from context.del of del modified by chardel0xff replace into ...
```

This will override the default load character string delimiter which is “0xff”.

Controlling DB2 Audit Facility Activities

As part of our discussion on the control of the audit facility activities, we will use a simple scenario: A user, *newton*, runs an application called *testapp* that connects and creates a table. This same application is used in each of the examples discussed below.

We begin by presenting an extreme example: You have determined to audit all successful and unsuccessful audit events, therefore you will configure the audit facility in the following way:

```
db2audit configure scope all status both
```

Note: This creates audit records for every possible auditable event. As a result, many records are written to the audit log and this reduces the performance of your database manager. This extreme case is shown here for demonstration purposes only; there is no recommendation that you configure the audit facility with the command shown above.

After beginning the audit facility with this configuration (using "db2audit start"), and then running the *testapp* application, the following records are generated and placed in the audit log. By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.10.555345;category=CONTEXT;
audit event=CONNECT;event correlator=2;database=FOO;
application id=*LOCAL.newton.980624124210;
application name=testapp;

timestamp=1998-06-24-08.42.10.944374;category=VALIDATE;
audit event=AUTHENTICATION;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;execution id=newton;
application id=*LOCAL.newton.980624124210;application name=testapp;
auth type=SERVER;

timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.561187;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;
```

```
timestamp=1998-06-24-08.42.11.594620;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.622984;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
object name=FOO;object type=DATABASE;access approval reason=DATABASE;
access attempted=CONNECT;

timestamp=1998-06-24-08.42.11.801554;category=CONTEXT;
audit event=COMMIT;event correlator=2;database=FOO;userid=boss;
authid=BOSS;application id=*LOCAL.newton.980624124210;
application name=testapp;

timestamp=1998-06-24-08.42.41.450975;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;object schema=NULLID;
object name=SQLC28A1;object type=PACKAGE;
access approval reason=OBJECT;access attempted=EXECUTE;
```

CREATE TABLE

```
timestamp=1998-06-24-08.42.41.476840;category=CONTEXT;
audit event=EXECUTE_IMMEDIATE;event correlator=3;database=FOO;
userid=boss;authid=BOSS;application id=*LOCAL.newton.980624124210;
application name=testapp;package schema=NULLID;package name=SQLC28A1;
package section=203;text=create table audit(c1 char(10), c2 integer);

timestamp=1998-06-24-08.42.41.539692;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object schema=BOSS;object name=AUDIT;object type=TABLE;
access approval reason=DATABASE;access attempted=CREATE;

timestamp=1998-06-24-08.42.41.570876;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object name=BOSS;object type=SCHEMA;access approval reason=DATABASE;
access attempted=CREATE;

timestamp=1998-06-24-08.42.41.957524;category=OBJMAINT;
audit event=CREATE_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object schema=BOSS;object name=AUDIT;object type=TABLE;
```

```
timestamp=1998-06-24-08.42.42.018900;category=CONTEXT;audit event=COMMIT;
event correlator=3;database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;
```

As you can see, there are a significant number of audit records generated from the audit configuration that requests the auditing of all possible audit events and types.

In most cases, you will configure the audit facility for a more restricted or focused view of the events you wish to audit. For example, you may want to only audit those events that fail. In this case, the audit facility could be configured as follows:

```
db2audit configure scope audit,checking,objmaint,secmaint,sysadmin,
   validate status failure
```

Note: This configuration is the initial audit configuration or the one that occurs when the audit configuration is reset.

After beginning the audit facility with this configuration, and then running the *testapp* application, the following records are generated and placed in the audit log. (And we assume *testapp* has not been run before.) By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.561187;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.594620;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;
```

CREATE TABLE

(none)

The are far fewer audit records generated from the audit configuration that requests the auditing of all possible audit events (except CONTEXT) but only when the event attempt fails. By changing the audit configuration you can control the type and nature of the audit records that are generated.

The audit facility can allow you to create audit records when those you want to audit have been successfully granted privileges on an object. In this case, you could configure the audit facility as follows:

```
db2audit configure scope checking status success
```

After beginning the audit facility with this configuration, and then running the *testapp* application, the following records are generated and placed in the audit log. (And we assume *testapp* has not been run before.) By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.11.622984;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;

timestamp=1998-06-24-08.42.41.450975;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;object schema=NULLID;
object name=SQLC28A1;object type=PACKAGE;
access approval reason=OBJECT;access attempted=EXECUTE;

timestamp=1998-06-24-08.42.41.539692;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object schema=BOSS;object name=AUDIT;object type=TABLE;
access approval reason=DATABASE;access attempted=CREATE;

timestamp=1998-06-24-08.42.41.570876;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object name=BOSS;object type=SCHEMA;access approval reason=DATABASE;
access attempted=CREATE;
```

CREATE TABLE

```
(none)
```

Chapter 8. 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: All of the information on these topics, and the comparable topics from the *Command Reference* and the *Administrative API Reference*, have be consolidated into the *Data Movement Utilities Guide and Reference*.

The *Data Movement Utilities Guide and Reference* is your primary, single source of information for these topics.

Chapter 9. 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 if 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
- Disaster Recovery Considerations
- Reducing the Impact of Media Failure
- Reducing the Impact of Transaction Failure
- System Clock Synchronization in a Partitioned Database System
- Crash Recovery
- Recovery Method: Version Recovery
- Recovery Method: Roll-Forward Recovery
- ADSTAR Distributed Storage Manager.

If you have tables that contain DATALINK columns, also refer to “DB2 Data Links Manager Considerations” on page 440.

One type of problem that requires point-in-time roll-forward recovery is the corruption of data that is caused by errant logic or incorrect input in an application. You can use 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 information 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*. *Crash recovery* automatically attempts to recover the database after a failure. There are two ways to recover a damaged database: *version recovery* and *roll-forward recovery*.

Non-recoverable databases have both the *logretain* and *userexit* database configuration parameters disabled. This means that the only logs that are kept are those required for crash recovery. These logs are known as *active logs*, and they contain current transaction data. Version recovery using *offline* backups is the primary means of recovery for problems with a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) When you restore the database, you can only restore it offline, and it is restored to the same state it was in when you took the backup image.

Recoverable databases have either the *logretain* database configuration parameter set to "RECOVERY"; or, *userexit* database configuration parameter enabled; or both. Active logs are still available for crash recovery, but you also have the *archived logs*, which contain committed transaction data. When you restore the database, you can only restore it offline, and it is restored to the same state it was in when you took the backup image. However, with forward recovery, you can then roll the database *forward* (that is, past the time of the backup image) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

Unlike non-recoverable databases, you can perform the backup operation for a recoverable database either offline or *online* (online meaning that other applications can connect to the database during the backup operation). The database restore and roll forward operations must always be performed offline. Although during an online backup operation, changes may also be occurring on the tables, roll-forward recovery ensures that *all* table changes are captured and reapplied if that backup is restored.

If you have a recoverable database, you can also back up, restore, and roll forward individual table spaces in it. When you back up a table space online, it is still available for use, and changes made to its tables during the backup are recorded in the logs. When you perform an online restore or roll forward on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

Crash recovery 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.

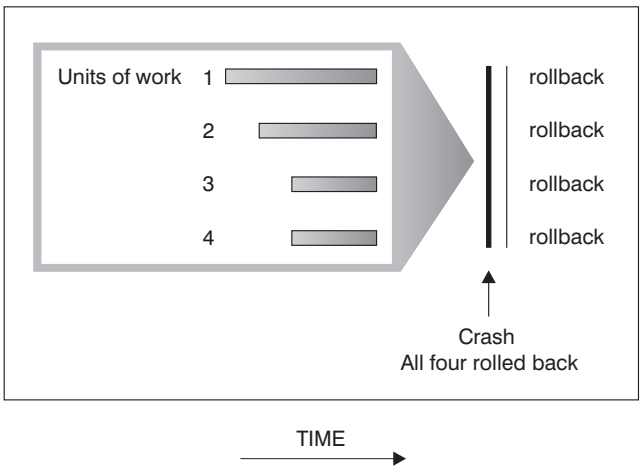


Figure 28. Rolling Back Units of Work

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.

When a database is in a consistent and usable state, it is known as a "point of consistency" for the database. It is a "snapshot" of the database at a point of time that saves a copy of the data in the database in a backup copy. An offline database backup is a point of consistency. After a point of consistency is reached, all transactions have been resolved and all data is free for use by other users or applications.

You can move to a point of consistency following a crash 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 moves the database to a consistent and usable state. If, however, crash recovery occurs for a database that is enabled for forward

recovery (that is, the *logretain* configuration parameter is set to "RECOVERY" or the *userexit* configuration parameter is enabled), and an error occurs during crash recovery that is attributable to an individual table space, that table space is taken offline, and cannot be accessed until it is repaired. Crash recovery continues. See "Rolling Forward Changes in a Table Space" on page 418 for more information. 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 offline as mentioned above. You may need to conduct a version recovery or a roll-forward recovery, depending on the error.

There are two recovery methods supported by DB2:

1. Version recovery allows for the restoration of a previous version or image of the database that was made using the BACKUP command.

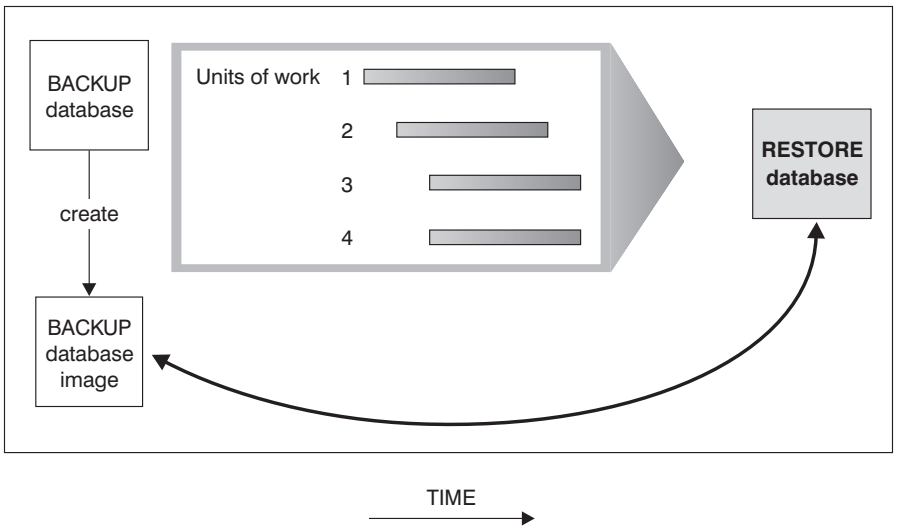


Figure 29. Restoring a Database

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. (To re-create these units of work requires the roll-forward recovery method, which is discussed later.)

Using the version 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*.

2. To use the roll-forward recovery method, you must have taken a backup of the database as well as archived the logs (by enabling either the *logretain* or *userexit* database configuration parameters, or both. For information on the decisions that you must make regarding the logging procedure that you use, see "Database Logs" on page 373.) If you restore the database and specify WITHOUT ROLLING FORWARD, it is the same as the version recovery method. The database is restored to a state identical to the time when the offline backup was made. If you restore the database and do *not* specify WITHOUT ROLLING FORWARD for the restore operation, the database will be in the roll-forward pending state at the end of the restore. This allows roll-forward recovery to take place.

   The two types of roll-forward recovery to consider are:

   a. *Database roll-forward recovery* 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 its state at a particular point in time, or to its state just before the failure (that is, to the end of the active logs.)

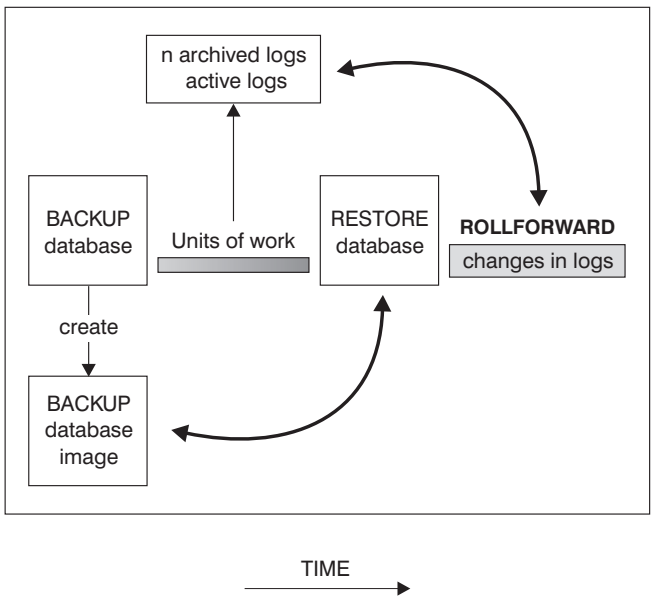


Figure 30. Database Roll-Forward Recovery

In a partitioned database system, the database is located across many database 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, restore, and roll-forward table spaces. To perform a *table space restore and roll-forward*, you need a backup image of either the entire database (that is, all of the table spaces) or of one or more individual table spaces; and, you need the log records that affect the table spaces to be recovered. You can apply the log records by rolling forward through the logs to one of two points:

   - To the end of the logs; or,
   - To a particular point in time (called *point-in-time* recovery).

   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 roll-forward recovery method with table spaces, 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.

Table space roll-forward recovery can be used in the following two situations:

a. After a table space restore, the table space is always in the roll-forward pending state, and it must be rolled forward. Again, use the ROLLFORWARD command to apply the logs against the table spaces to either a point in time, or to the end of the logs.
b. If one or more table spaces are in a *roll-forward pending* state after crash recovery, first correct the problem with the table space. In some cases, correcting the problem with the table space does not involve performing a RESTORE. For example, a power loss could leave the table space in a roll-forward pending state. If the problem is corrected before crash recovery, crash recovery may be sufficient to take the database to a consistent, usable state. No RESTORE is required in this case. Once the problem with the table space is corrected, you can use the ROLLFORWARD command to apply the logs against the table spaces to either a point in time, or to the end of the logs.

   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.

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 roll-forward request to all database partitions. This means the table space must be restored on all database partitions on which the table space resides.

In a partitioned database system, if you are rolling forward a table space *to the end of the logs*, you have to supply the list of nodes if you do *not* want to roll the table space forward on all database partitions. If you want to roll forward all table spaces on all database partitions that are in the roll-forward pending state 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 backups?

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
- Restrictions on Using Different Operating Systems
- Damaged Table Space Recovery
- Recovery Performance Considerations.

While the general focus of this chapter is on the database, your overall recovery planning should also 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)

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.

Active logs are used by crash recovery to prevent a failure (system power, application error) from leaving a database in an inconsistent state. 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. During crash recovery, changes recorded in these logs that were made to the data but not committed because of the failure are rolled back. Changes that were committed but were not physically written from memory (buffer pool) to disk (database containers) are redone. These actions ensure the integrity of the database. 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.

Archived logs are used specifically for roll-forward recovery. They can be:

online archived logs
When all changes in the active log are no longer needed for

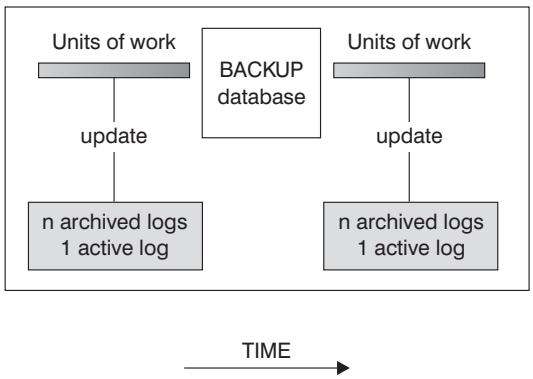
normal processing, the log is closed, and becomes an archived log. An archived log is said to be *online* when it is stored in the database log path directory.

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 G. User Exit for Database Recovery" on page 733 for additional information.) An archived log is said to be *offline* when it is no longer found in the database log path directory.

Roll-forward recovery can use both archived logs and active logs 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 committed 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 by re-applying committed updates in both archived and active logs. 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 31. Active and Archived Database Logs in Roll-forward Recovery

Two database configuration parameters 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. Refer to *Administration Guide, Performance* for more information on these configuration parameters.

To determine which log extents in the database log path directory are archived logs, check the value of the database configuration parameter *loghead*. This parameter indicates the lowest numbered log that is active. Those logs

with sequence numbers less than *loghead* are archived logs and can be moved. You can check the value of this parameter by using the Control Center; or, by using the command line processor and the GET DATABASE CONFIGURATION command to view the "First active log file". Refer to *Administration Guide, Performance* for more information on this configuration parameter.

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 creates the table will not be logged. This not only reduces the logging that is done, but also obtains better performance for your application. You can also obtain the same behavior for existing tables by using the ALTER TABLE statement with the NOT LOGGED INITIALLY parameter.

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 must be flushed out to disk 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. Refer to "Concurrency" in the *Administration Guide, Performance* for more information.

Refer to the *SQL Reference* for more information about creating tables.

Point of Recovery

The version and roll-forward recovery methods provide different points of recovery. The version method involves making an offline, full database backup copy of the database at scheduled times. With this method, the recovered database is only as current as the backup copy that was restored. 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 418.

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, even 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 not overwriting backups and logs, saving more than one full database backup and its associated logs as an extra precaution.

You may have an active database where more logging occurs and more archived logs are created. If the amount of time needed to apply archived logs when recovering and rolling forward a database is a major concern, then you need to consider the cost of having more frequent database backups. More frequent database backups reduces the number of archived logs you need to apply when rolling forward through the archived logs.

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.

Notes:

1. You can only use an online backup if you have the database log (or logs) that span the time that backup operation both started and completed.
2. offline backups are faster than online backups.

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. 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. 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.

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.

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.

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 roll-forward 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.

Note: The setting of the enable intra-partition parallelism (*intra_parallel*) database manager configuration parameter does not affect the performance of either backup or restore operations. Multiple processes will be used for both of these operations, regardless of the setting of the *intra_parallel* parameter.

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 version recovery method requires space to hold the backup copy of the database and the restored database. The roll-forward recovery 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 an 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 1 GB SMS table space (with LOBs), you will need more than 1 GB 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 413.)

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 are 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 after they are no longer in the active set of logs.
3. Use a user exit program to copy these logs to another storage device in your environment. (See "Appendix G. User Exit for Database Recovery" on page 733 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 G. User Exit for Database Recovery" on page 733 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.

Restrictions on Using Different Operating Systems

When working in an environment that has more than one operating system, you must consider that the backup and recovery plans cannot be integrated. That is, you may not use the BACKUP DATABASE command on one operating system and the RESTORE DATABASE command on another operating system. You should keep the recovery plans for each operating system separate and independent.

If you must move tables from one operating system to another, use the *db2move* command; or, use the EXPORT with the IMPORT or LOAD commands.

Damaged Table Space Recovery

A damaged table space is a table space that has one or more containers which cannot be accessed. When a table space is damaged, it is often caused by media problems that are either permanent (for example, a bad disk), or temporary (for example, an offline disk, or an unmounted file system).

If the damaged table space is the system catalog table space, the database cannot be restarted. If the container problems cannot be fixed leaving the original data intact, then the only options are:

- To restore the database; or,
- To restore the catalog table space.

  Note: Table space restore is only valid for recoverable databases since the database must be rolled forward.

If the damaged table space is not the system catalog table space, then DB2 attempts to make as much of the database available as possible. The overall success of this attempt is dependent on the logging strategy.

If the damaged table space is a temporary table space and it is the only one existing, then as soon as a connection to the database is made you should create a new temporary table space. Once created, the new temporary table space can be used and normal database operations requiring temporary table space can resume. You can then, if you wish, drop the OFFLINE temporary table space. There are special considerations regarding a table reorganization using a temporary table space:

- If the database or database manager configuration parameter *indexrec* is set to “RESTART”, all invalid indexes must be rebuilt during database activation — this includes indexes from reorganization that crashed during the build phase.
- If there are incomplete reorganization requests in a damaged temporary table space then you may have to set the *indexrec* configuration parameter to “ACCESS” to avoid restart failures.

Table Space Recovery: Recoverable Databases

The damaged table space is put in an OFFLINE state and a ROLLFORWARD PENDING state since crash recovery is necessary. The restart will succeed assuming there is no additional problem. The damaged table space can be used again once the user either:

1. Fixes the damaged containers without losing the original data and then does a table space ROLLFORWARD. (The roll forward will first attempt to bring it from an OFFLINE to an ONLINE state.)
2. Does a table space RESTORE after fixing the damaged containers (with or without losing the original data) and then a roll forward.

Table Space Recovery: Non-recoverable Databases

Since crash recovery is necessary, and logs are not kept indefinitely, the restart can only succeed if the user is willing to drop the damaged table spaces. (Successful completion of recovery means that the log records necessary to recover the damaged table space(s) to a consistent state will be gone — thus the only valid action on such table space(s) would be to drop them.)

The user can do this by:

Issuing an unqualified restart.

It will succeed if there are no damaged table spaces. If it fails with `SQL0290N Table space access is not allowed`, the user can then look in the db2diag.log for a complete list of table spaces that are currently damaged.

- If the user is willing to drop all these table spaces once the restart is complete, he can then issue a restart listing all the damaged table spaces in the DROP PENDING TABLESPACES. As restart tries to bring up a table space, if any table space is damaged it looks to see if that table space is included in the DROP PENDING TABLESPACE list:
  - If it is: The table space is put into DROP PENDING and the users only allowed action on the table space after recovery is a ’DROP TABLESPACE’, restart continues but without recovering this table space
  - If it is not: the restart fails with SQL0290N.

- If the user is unwilling to drop (and thus lose the data in) these table spaces; his options are to:
  1. Wait and fix the damaged containers (without losing the original data) and then trying the restart again
  2. Perform a database restore.

Note: Putting a table space name into the DROP PENDING TABLESPACE list does not mean the table space will be in DROP PENDING state. This is true only if the table space is damaged during the restart time. Once the restart is successful the user should issue ″DROP TABLESPACE″ statements to drop each of the table spaces that are in DROP_PENDING (do a LIST TABLESPACES if unsure). This way the space can be reclaimed and or the table spaces can be re-created.

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.

    A recommended number of buffers is the number of media devices or containers plus the number used with the PARALLELISM option plus a few extra.
  - Use multiple source devices.

  - Set the PARALLELISM option for the restore operation to be at least one (1) greater than the number of source devices.
- 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. 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. 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 LOB data can be reproduced from a separate source, when creating or altering a table to include LOB columns, choose the NOT LOGGED option. If you choose not to restore the table spaces that contain long field and LOB data, but you need to restore 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.
- The following apply for both backup and restore operations:
  - 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. Typically, disaster recovery requires that you restore the entire database, so when a major disaster occurs, a full database backup is needed on a standby site (even if you have a separate backup image of every table space, you cannot use them to recover the database). 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 403.

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.

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. Refer to *Administration Guide, Performance* for more information on setting *max_time_diff*.
- 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.

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
- Identifying the Failed Database Partition Server.

Getting to a Consistent Database

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.

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 were not complete. Also, some committed units of work may not have been flushed to disk. The partially completed (or nonflushed) units of work leave the database in an inconsistent, or unusable, state.

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 automatically 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". Refer to *Administration Guide, Performance* for more information on this parameter.

Transaction Failure Recovery in a Partitioned Database Environment

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). 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" on page 391, and "Transaction Failure Recovery on the Failed Database Partition Server" on page 392.

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 478.

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 acknowledgment of the COMMIT to the coordinator node. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments 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 on the active database partition servers, and the coordinator node is not aware of this (because it is not available). See “Recovering from Problems During Two-Phase Commit” on page 481 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. The coordinator node will either send a rollback or a disconnect message to the other database partition servers. The transaction will only be indoubt on database partition servers that are still active if the coordinator node returns SQL0279.

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, refer to the START DATABASE MANAGER command and API in the *Command Reference* and *Administrative API Reference* 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 acknowledgments 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 held 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, refer to the LIST INDOUBT TRANSACTIONS command and API in the *Command Reference* and *Administrative API Reference* manuals respectively.

Note: The LIST INDOUBT TRANSACTIONS command is also used for transactions in a distributed transaction environment. See "Chapter 10. Distributed Databases" on page 465 and "Chapter 11. Using DB2 with an XA-Compliant Transaction Manager" on page 489 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 Enterprise - Extended 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: Version Recovery

Version recovery using the BACKUP command in conjunction with the RESTORE command puts the database in a state that has been previously saved. You use this recovery method with non-recoverable databases (that is, databases for which you do not have archived logs). You can also use this method with recoverable databases by using the WITHOUT ROLLING FORWARD option.

In this section, 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.

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.

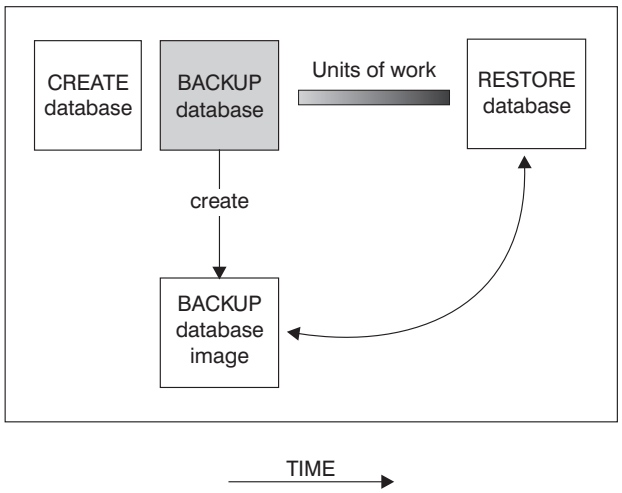


Figure 32. Creating a Database Image

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. Because this recovery method does not support roll-forward recovery, regularly back up the database on this list of nodes.

In a distributed request system, the BACKUP and RESTORE commands apply to the distributed request database and the metadata stored within that database catalog (wrappers, servers, nicknames, etc.). Data source objects (tables and views) are not backed-up or restored unless those objects are stored in the distributed request database.

You must keep in mind the recovery method to be used. The following sections provide 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 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 452 for information on ADSM.

  Under OS/2, you can also back up to diskette or to a user exit.

  Note: In OS/2, when backing up a database online to a user exit, 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.

- 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.

  Note: If you use a variable block size with your tape devices, ensure that the DB2 buffer size is either less than or equal to the maximum variable block size that the device is configured for. Otherwise, the backup will succeed but the resulting image is not guaranteed to be recoverable.

- Multiple files may be created to contain the backed up data from the database.
- 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.

If you have tables that contain DATALINK columns, also see "Backup Utility Considerations" on page 442.

To use tape devices, DB2 users on SCO UnixWare 7 must specify BUFFER to be 16. The default value of BUFFER is 1024 pages. If BUFFER is set to zero, the database manager configuration parameter **BACKBUFSZ** must be set to 16.

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:
  - Increase the value of the PARALLELISM parameter.

    Using this parameter can dramatically reduce the amount of time required to complete the backup. The PARALLELISM parameter defines the number of processes or threads that are started to read data from the database. Each process or thread is assigned to back up a specific table space. When it completes backing up the table space, it requests another. You should note, however, that each process or thread requires both memory and CPU overhead: for a heavily loaded system, you should leave the PARALLELISM parameter at its default value of 1.
  - 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.

Refer to *Administration Guide, Performance* for more information on this configuration parameter.

- Use multiple target devices.

- 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.
- 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 is in a partially restored state due to a system crash during any stage of restoring the database, you must successfully restore the database before you can back it up.
  - 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 B. Planning Database Migration" on page 641, 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, unless it is in the backup pending state.
- 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. 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 435.

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.

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*`.cat`*nnnn*`.timestamp.number`

On other operating systems, the format would appear as:

```
Database alias.Type\Instance name.nodennn\catnnnn\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.

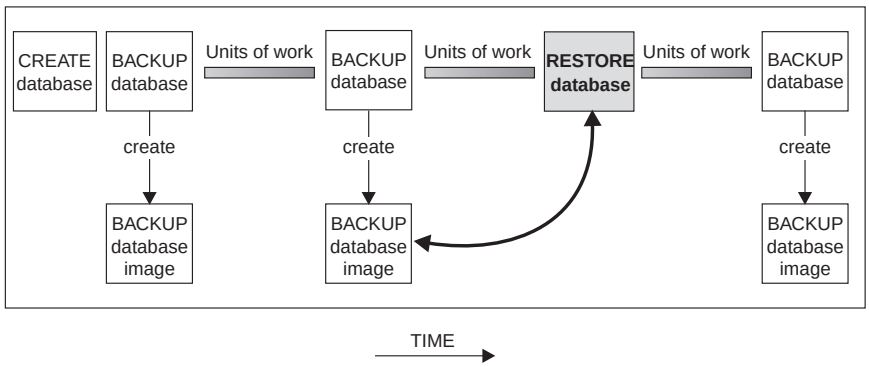


Figure 33. Restoring a Database Using a Backup Image

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 backup. To restore to a new database, you must have SYSADM or SYSCTRL authority.
- You can only use this command if the database has been previously backed up with the BACKUP command.
- If you use the Control Center, you cannot restore backups that were taken previous to the current version of DB2.
- In OS/2, the RESTORE command can call a user exit program only if a user exit program was used to backup the database.
- 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 restore of only the recovery history file
- 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 452.)
- 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.
- The database may be local or remote.
- If the WITHOUT DATALINK option is not specified, and the DB2 Data Links Manager containing the DATALINK data is not available, then the restore operation will fail.

If the option is specified and the DB2 Data Links Manager containing the DATALINK data is not available, all table spaces containing tables with DATALINK values on the unavailable server are placed in the RESTORE PENDING state. These table spaces must be restored again when the Data Links server becomes available.

- To use tape devices, DB2 users on SCO UnixWare 7 must specify BUFFER to be 16. The default value of BUFFER is 1024 pages. If BUFFER is set to zero, the database manager configuration parameter **BACKBUFSZ** must be set to 16.

If you have tables that contain DATALINK columns, see both "Restore and Rollforward Utility Considerations" on page 442 and "Restoring Databases from an offline Backup without Rolling Forward" on page 444.

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.
- 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.

  Refer to *Administration Guide, Performance* for more information on this configuration parameter.
- 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 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 452 for information on ADSM.

  If you use ADSM and do not specify the TAKEN AT parameter, ADSM retrieves the latest backup copy.

  Under OS/2, the backup copy of the database could also be located on diskette or through a user exit.

  Under Windows 95 and Windows NT, the backup copy of the database could also be located on diskette.

- Once the RESTORE command starts, the database is not usable until the RESTORE command 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 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, 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.

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 to an existing database. 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.

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. 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.

Note: 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.

When you restore to a new database, 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 Method: Roll-Forward Recovery

Roll-forward recovery using the BACKUP command in conjunction with the RESTORE and ROLLFORWARD commands allows the database or table space to be recovered to its state at a specified point in time.

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 version recovery 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 having either the *logretain* database configuration parameter set to ″RECOVERY″; or the *userexit* database configuration parameter is enabled; or both. When either of these parameters are configured as described in the previous sentence, the database is enabled for *roll-forward recovery*.

When the database is recoverable, you can perform backup, restore, and roll-forward recovery at both the database and the table space level. The backups of the database and table space can be online. online restore and rollforward are also available at the table space level.

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 400. 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 418

For more information about the database configuration parameters associated with logging, see “Configuration Parameters for Database Logging” on page 414.

Backup Considerations

Following are the backup considerations that apply when your database is enabled for forward recovery. For general information that applies to performing backups, refer to the following:

- “Backing Up a Database” on page 394
- “Planning to Use the BACKUP Command” on page 396
- “Invoking the BACKUP Command” on page 397
- “Backup Images Created by BACKUP” on page 398.
- Roll-forward recovery is not enabled by the default setting (“No”) of the *logretain* and *userexit* configuration parameters. The default for both parameters is set to “No” 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.
- 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.
  - If any of the table spaces in a database is in an “abnormal” state, you cannot back up the database or that table space, unless it is in the backup pending state.
- 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 452 for information on ADSM.

  Under OS/2, you can also back up to diskette or to a user exit.

- 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.
- To reduce the amount of time required to complete a backup:
  - Use table space backups.

    You can back up (and subsequently recover) part of a database 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.
  - Increase the value of the PARALLELISM parameter so that it reflects the number of table spaces that are being backed up.
- The considerations for backing up table spaces are as follows:
  - 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.
  - If you have tables that span more than one table space, you should backup (and restore) the set of table spaces together.
  - 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 can still use the other table spaces in the database, unless the table space in this state has the system catalog tables. In this situation, you cannot 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.
- The considerations for a partitioned database environment are as follows:

  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.

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 435.

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.

Restore Considerations

Following are the restore considerations that apply when your database is enabled for forward recovery. For general information that applies to performing restores, refer to the following:

- "Restoring a Database" on page 400
- "Planning to Use the RESTORE Command" on page 401
- "Invoking the RESTORE Command" on page 402
- "Redefining Table Space Containers During RESTORE" on page 403
- "Restoring to an Existing Database" on page 404
- "Restoring to a New Database" on page 405.

The items you should consider are:

- You can restore a backup copy of a full database backup or table space backup to an existing database. 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.
- When you restore to an existing database, and the database seeds are the same, the logs are retained.
- You can only use the RESTORE command if the database or table space has been previously backed up with the BACKUP command.
- After a database enabled for roll-forward recovery is restored, it is in the roll-forward pending state. The database is unusable until it is rolled forward. The exception occurs when a restore WITHOUT ROLLING FORWARD is specified. You cannot turn roll-forward off if an online database backup was restored or only selected table space backups were restored.
- 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 452 for information on ADSM.

  If you use ADSM and do not specify the TAKEN AT parameter, ADSM retrieves the latest backup copy.

  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.
  – 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.

  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.

- The considerations for restoring table spaces are as follows:

  – 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.)
  – You cannot restore a table space backup to a new database.
  – If you backed up tables that spanned more than one table space, you should restore the set of table spaces together.
  – 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.

- A table space restore can be online (share mode) or offline (exclusive mode).
- 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.
- You cannot perform an online table space restore of the system catalog tables.
- 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.
- In OS/2, a partial or subset restore is not possible when restoring from a user exit.

• The considerations for redirected restore are as follows:
- 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. 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.
- 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. In fact, 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. (You can edit the list of table space containers on the CONTAINERS - CHANGE window of the Restore Database or Restore Table Space notebook in the Control Center.)

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.

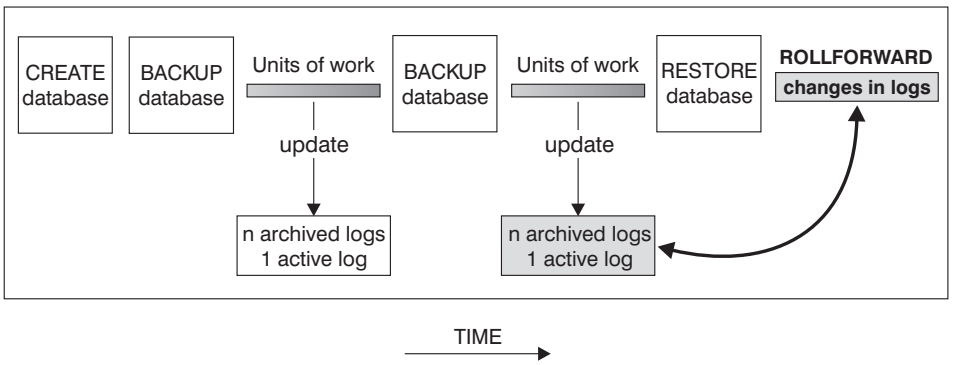


Figure 34. Roll-Forward Recovery

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 418.

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.

If you have tables that contain DATALINK columns, also see “Restoring Databases and Table Spaces and Rolling Forward to the End of the Logs” on page 445 and “Restoring Databases and Table Spaces and Rolling Forward to a Point in Time” on page 445.

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. Refer to *Administration Guide, Performance* for more information on configuring DB2 UDB.

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.

The total log file size limit is 4 GB. That is, the number of logfiles (LOGPRIMARY + LOGSECOND) multiplied by the size of each logfile in bytes (LOGFILSIZ * 4096) must be less than 4 GB.

Refer to *Administration Guide, Performance* for more information on this configuration parameter.

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.

Refer to *Administration Guide, Performance* for more information on this configuration parameter.

Log size (logfilsiz)

This parameter determines the number of pages for each of the configured logs. A page is 4 KB in size.

Note: The total log file size limit is 4 GB (that is, (*logfilsiz* + *logprimary*) x *logfilsiz* < 4 GB / 4096).

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 382), 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 G. User Exit for Database Recovery" on page 733 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)

The database logs are initially created in SQLOGDIR, which is a subdirectory of the database directory. You can change the location where active logs and future archive logs are placed by changing the value for this configuration parameter to point to either a different directory, or to a device. Archive logs that are currently stored in the

database log path directory are not moved to the new location if the database is configured for roll-forward recovery.

Because you can change the log path location, the logs needed for roll-forward recovery may exist in different directories or on different devices. You can change this configuration parameter during the roll-forward process to allow you to access logs in multiple locations.

The change to the value of *newlogpath* will not be applied until the database is in a consistent state. A database configuration parameter indicates the status of the database. Refer to *Administration Guide, Performance* for more information on the *database_consistent* status indicator. See "Considerations for Managing Log Files" on page 429 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 by setting it to "RECOVERY" 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 G. User Exit for Database Recovery" on page 733, 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 enabled, the user exit is called to archive log files away from the active log path. When the *logretain* configuration parameter is set to "RECOVERY", 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 instead of using the entire 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, and requires less time than backing up the entire database.

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.

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.

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.

  The table space could also be in the *roll-forward-in-progress* state if the roll forward operation did not complete, or AND STOP was not specified for the operation.
- 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 before it can be used.

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 you do not want to roll the table space forward, you can specify ROLLFORWARD STOP, which is the same as rolling the table space forward to the time of the restored backup.

  Note: You cannot do this if the backup image was taken online. In this situation you must roll forward to at least the end of the backup.
- If you are rolling forward to a point in time, and 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 minimum recovery time is updated when DDL statements are executed against the table space, or against tables in the table space. The table space must be rolled forward to at least the minimum recovery time so that is synchronized with the information in the system catalog tables.
- It is possible to recover the data from tables that have been accidentally dropped, give certain conditions. See “Dropped Table Recovery” on page 427 for more information.
- 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 recovery time to determine a desirable time to stop the roll forward.

- 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.
- If you want to roll forward a table space to a point in time and a table in the table space is either of the following:
  - An underlying table for a summary table that is in another table space
  - A summary table for a table in another table space

  You should roll forward both table spaces to the same point in time. If you do not, the summary table is placed in the *check pending* state at the end of the 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 point in time to stop rolling forward 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 all updates made to it between the point in time that you rolled forward to and the current time have been removed. You can no longer roll forward the table space to the current time from a previous database or table space backup. 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.)

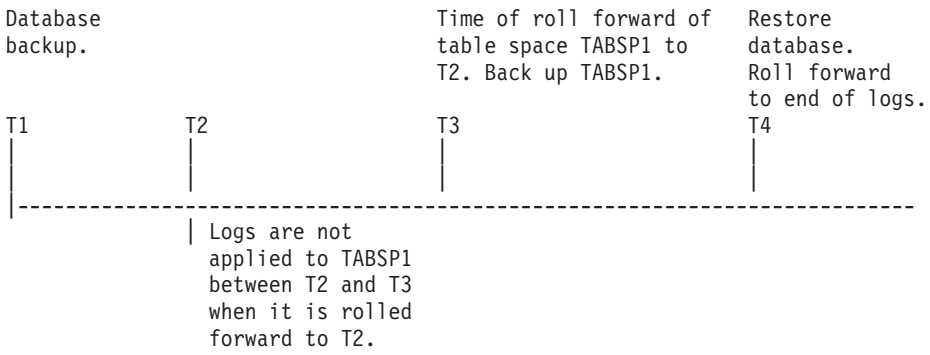


Figure 35. Table Space Backup Requirement

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 it is rolled forward to a point in time allows you to roll that table space forward past a previous 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.

1. Restore the database.
2. Restore the table space.
3. Roll forward the database.
4. Roll forward the table space.

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.

If you cannot find the backup 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. You do not need to restore the table space again because it was restored from the database backup.
- 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.

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 across database partitions.

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.
- In a partitioned database environment, the rollforward must be issued from the catalog node of the database.
- 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 during a table space rollforward, or by specifying STOP on the ROLLFORWARD command). You can, however, perform an online roll forward of table spaces as long as SYSCATSPACE is not included. 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:

1. Issue ROLLFORWARD (without the STOP option).
2. Issue ROLLFORWARD QUERY STATUS.

   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.
3. Issue ROLLFORWARD STOP. After a ROLLFORWARD STOP, it is not possible to roll forward additional changes.

- You can perform a partial or subset restore of a backup created using the current version of DB2. This cannot be done with earlier versions of DB2.
- 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 offline because of a media error.

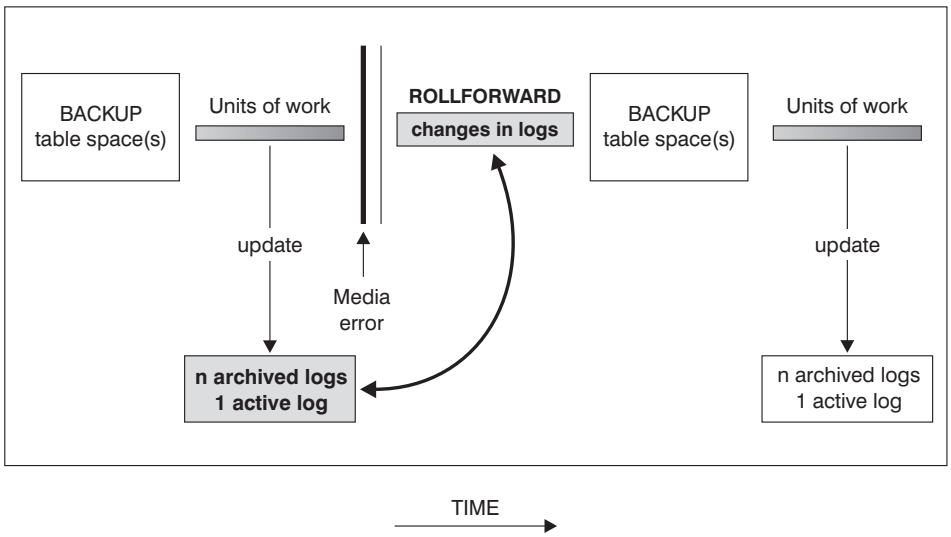


Figure 36. Table Space Roll-forward Recovery

- You do not have to recover your database 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 cancel a roll-forward operation (that is, ROLLFORWARD STOP was not specified, or the ROLLFORWARD command failed) 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 and you specify a list of table spaces that are in the roll-forward pending state, they are put in the restore pending state. In this situation, there is no roll forward command in progress.

  Note: If no table space list is specified, SQL4906 is issued.

– If multiple table spaces are being rolled forward to the end of the logs and you specify ROLLFORWARD CANCEL with a list, only 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 specify ROLLFORWARD CANCEL without a list, all table spaces that are in the rollforward-in-progress state are put in the restore pending state and the ROLLFORWARD command is no longer in progress.

– If you issue ROLLFORWARD CANCEL and one or more table spaces are being rolled forward to a point in time, they are all put in the restore pending state, whether you specify a list or not. Even if you specify a list, the list is ignored and all table spaces that are in the roll-forward-in-progress state are put in the restore pending state and the ROLLFORWARD command is no longer in progress.

Note: You cannot use ROLLFORWARD CANCEL to cancel a roll-forward operation that is running. You can only use it to cancel a roll-forward operation that completed but did not have ROLLFORWARD STOP issued for it, or for a roll-forward operation that failed before completing.

If you have tables that contain DATALINK columns, also see “Restore and Rollforward Utility Considerations” on page 442.

You cannot roll forward a partitioned database from a Version 2 client.

Invoking the ROLLFORWARD Command

There are a number of considerations before invoking the ROLLFORWARD command:

• When you invoke the ROLLFORWARD command, you can specify a time 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, which is the last update to the system catalogs for this table space, or its tables. You can get the minimum recovery time for a table space using the LIST TABLESPACES SHOW DETAIL command. Refer to the *Command Reference* for details on this command.
- 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. If a table space is being rolled forward, it is left in the rollforward-in-progress 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.

3. If an entry in the file identified by the DB2LOADREC environment variable is not valid, then the old load copy location file will be used to provide information to replace the invalid entry.

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.
- If there is even one of the entries in the location file that is not valid, then the previous load copy location file is used to provide those entries.
- 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 452.

- 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.

Refer to *Data Movement Utilities Guide and Reference* for more information on LOAD.

Dropped Table Recovery

There may be times when you have accidentally dropped one or more tables whose data you still need. If you have such tables where the data must not be lost, even accidentally, you should consider making that table recoverable following a drop.

You can recover the table's data by using a database RESTORE followed by a database roll-forward. This may be time consuming if the database is large, and will make your data unavailable during the recovery. By utilizing the dropped table recovery you can recover your dropped table's data using table space level restore and roll forward. This, in turn, will be faster than a database level recovery and will allow your database to remain available to the users.

For a dropped table to be recoverable, the table space in which the table resides must have its DROPPED TABLE RECOVERY option turned on. This can be done using the ALTER TABLESPACE statement, or during the CREATE TABLESPACE statement. Refer to the *SQL Reference* for more information on these statements.

The DROPPED TABLE RECOVERY option is table space specific. To determine if a table space has this characteristic, you can query the DROP_RECOVERY column of the table space name in the *syscat.tablespaces* catalog table.

When a DROP TABLE statement is run against a table that is in a table space having the DROPPED TABLE RECOVERY option "ON", an additional log entry is made in the log files. The log entry has information identifying the dropped table. An entry is also made in the recovery history file and contains information that can be used to re-create the table.

You can recover a dropped table by doing the following:

1. Obtain the identification of the dropped table. This identification can be found in the recovery history file by using the LIST HISTORY DROPPED TABLE command. A list of tables that have been dropped is displayed and the information needed to re-create the table. Refer to the *Command Reference* for more information on this command.
2. Restore a database or table space level backup taken before the table was dropped.
3. Roll forward to a point in time after the drop using the RECOVER DROPPED TABLE option on the ROLLFORWARD command. Other information that is required when using this option includes the dropped table identification, and the directory path where the output files will be written. The directory path must either be accessible by all database partitions, or exist on each partition. Refer to the *Command Reference* for more information on this command.
4. Re-create the table using the CREATE TABLE statement from the recovery history file.
5. Import the data exported by the ROLLFORWARD command into the table.

The exported data is written to files using the following naming convention: Under the *export_directory* specified by the user in the ROLLFORWARD command, a subdirectory is created by each database partition. The user may create the subdirectories before the roll forward request is issued. This may be used by you to export the data to a particular drive or machine. The subdirectories are named "NODE*nnnn*", where *nnnn* is the database partition or node number. In each subdirectory a data file is exported under the name "data". The data files contain the dropped table's data as it existed on each database partition.

There are some restrictions on the type of data that is recoverable from the dropped table. Only a single dropped table can be recovered at a time. To recover several dropped tables, the recovery sequence presented above must be carried out each time another table is to be recovered. It is not possible to recover:

- LOB or LONG data. The DROPPED TABLE RECOVERY option is not supported for LONG table spaces. Attempts to use it for a LONG table space will return error SQL628N. If an attempts to recover a dropped table that contains LOB or LONG VARCHAR columns, these columns will be set to NULL in the generated export file. The DROPPED TABLE RECOVERY option should only be turned "ON" for REGULAR table spaces and not TEMPORARY or LONG table spaces.
- The names of the linked files associated with DATALINK columns can be recovered. After importing the data, the table should be reconciled with the DB2 Data Links Manager. Backups of the files may or may not be restored by the DB2 Data Links Manager, depending on whether garbage collection has already deleted them.
- The meta information associated with rowtypes. (The data is recovered, but not the metadata.) The data in the hierarchy table of the typed table will be recovered. This data may contain more information than appeared in the typed table which was dropped.

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 that was used by roll-forward 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. Ensure you make a copy of the logs before executing the ROLLFORWARD command. (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 log path. The log path defaults to the SQLOGDIR subdirectory but can be changed with the *newlogpath* configuration parameter. To place them elsewhere, enable the database for user exit, or change the log path with *newlogpath*. In this case, you may need to 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 G. User Exit for Database Recovery” on page 733 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 37 on page 431 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.

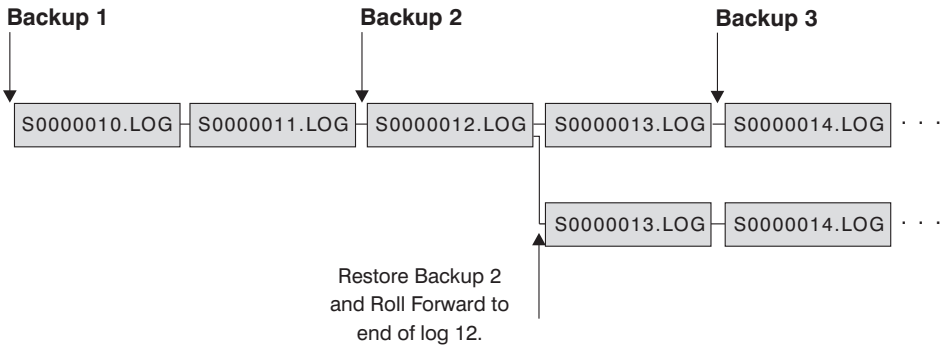


Figure 37. Reusing Log File Names

- 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.

Using Raw Logs

You can use a raw device for your database log. There are both advantages and disadvantages in doing so.

- The advantages are:
  - You can attach more than 26 physical drives to a system.
  - The file I/O path length is shorter. This may improve performance on your system. You should conduct benchmarks to evaluate if there are measurable benefits for your work load.
- The disadvantages are:
  - The device cannot be shared by other applications; the entire device *must* be assigned to DB2.
  - The device cannot be operated upon by any operating system utility or third-party tool which would backup or copy from the device.
  - You can easily wipe out the file system on an existing drive if you specify the wrong physical drive number.

You can configure a raw log with the *newlogpath* database configuration parameter. See "RAW I/O" on page 156 for an example of the syntax used to specify a raw device. Before doing so, however, consider the advantages and disadvantages listed above, and the additional considerations listed below:

- Only one device is allowed. You can define the device over multiple disks at the operating system level. DB2 will make an operating system call to determine the size of the device in 4 KB pages.

  If you use multiple disks, this will provide a larger device, and the striping that results can improve performance by faster I/O throughput.
- DB2 will attempt to write to the last 4 KB page of the device. If the device size is greater than 2 GB, the attempt to write to the last page will fail on operating systems that do not provide support for devices larger than 2 GB. In this situation, DB2 will attempt to use all pages, up to the supported limit.

  Information about the size of the device is used to indicate the size of the device (in 4 KB pages) available to DB2 under the support of the operating system. The amount of disk space that DB2 can write to is referred to as the *device-size-available*.

  The first 4 KB page of the device is not used by DB2 (this space is generally used by operating system for other purposes.) This means that the total space available to DB2 is *device-size = device-size-available - 1*.

- The *logsecond* parameter is not used. DB2 will not allocate secondary logs. The size of active log space is the number of 4 KB pages that result from *logprimary* x *logfilsiz*.
- Log records are still grouped into log extents, each with a log file size (*logfilsiz*) of 4 KB pages. Log extents are placed in the raw device, one after another. Each extent also consists of an extra two pages for the extent header. This means that the *number of available log extents* the device can support is *device-size* / (*logfilsiz*+ 2)
- The device must be large enough to support the active log space. That is, the *number of available log extents* must be greater than (or equal to) the value specified for the *logprimary* configuration parameter.
- If you are using circular logging, the *logprimary* configuration parameter will determine the number of log extents that are written to the device. This may result in unused space on the device.
- If you are using log retention (*logretain*) without a user exit, after the *number of available log extents* are all used up, all operations that result in an update will receive a log full error. At this time, you must shut down the database and take an offline backup of it to ensure recoverability. After the database backup, the log records written to the device are lost. This means that you cannot use an earlier database backup to restore the database, then roll it forward. If you take a database backup before the *number of available log extents* are all used up, you can restore and roll forward the database.
- If you are using log retention (*logretain*) with a user exit, the user exit program is called for each log extent as it is filled with log records. The user exit program must be able to read the device, and to store the archived log as a file. DB2 will not call a user exit to retrieve log files to a raw device. Instead, during roll forward recovery, DB2 will read the extent headers to determine if the raw device contains the log file to be used. If the required log file is not found in the raw device, DB2 will search the overflow log path. If the log file is still not found, DB2 will call the user exit to retrieve the log file into the overflow log path. If you do not specify an overflow log path for the **rollforward** command, DB2 will not call the user exit to retrieve the log during the roll-forward operations. For additional information about the calling the user exit program, see “Calling Format for UNIX-Based or Windows NT Operating Systems” on page 738.
- If you are using DPropR and writing logs to a raw device, the read log API will not call the user exit to retrieve log files. Requested log records, however, will be still be returned if they are available on the device. If you request logs that pre-date the oldest ones on the device, they will not be returned (the behavior is similar to DB2 not being able to find the log file that contains the requested log records).

Notes:

1. It is recommended that you do not use DPropR when you use a raw device for logging.
2. If you use the sqlurlog API, you should not use a raw device for logging.

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-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.

Recovery History File Information

A recovery history file is created with each database and is automatically updated whenever there is a:

- Back up of a database or table space
- Restore of a database or table space
- Roll forward of a database or table space
- Alter of a table space
- Quiesce of a table space
- Load of a table
- Drop a table
- Reorganization of a table
- Update of table statistics.

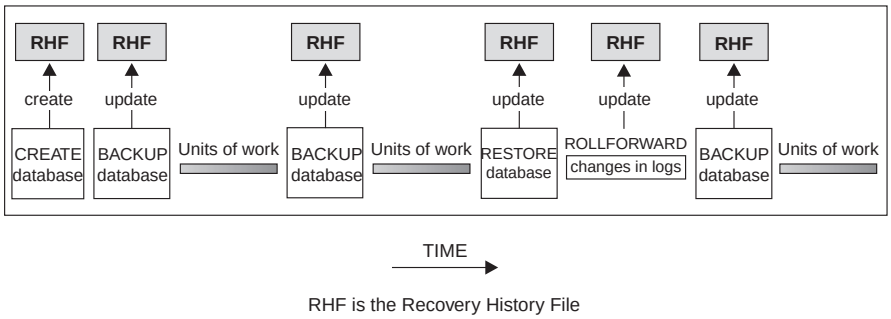


Figure 38. Creating and Updating the Recovery History File

You can use the summarized backup information in this file to recover *all* or *part* of the database to a given point in time. The information in the file includes:

- An identification (ID) field associated with each entry to uniquely identify that entry
- 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

- The status of the backup: active, inactive, expired, or deleted
- The last log sequence number saved by the database backup or processed by a roll-forward recovery.

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 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. Refer to *Administration Guide, Performance* for more information on this configuration parameter.

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 basic queries and commands are: OPEN, CLOSE, GET NEXT, UPDATE, and PRUNE. (For more information on the command syntax refer to the *Command Reference*. For more information on the API function call, refer to the *Administrative API Reference*. For more information on the Control Center, access the Control Center from your workstation.)

Detailed information about the history file is recorded in the SQLUHINFO structure. For more information about this structure, refer to the *Administrative API Reference*.

Garbage Collection

The number of DB2 database backups documented in the Recovery History File is monitored automatically by something called “DB2 Garbage Collection”. The configuration parameter *num_db_backups* defines how many ″active″ backups are kept. An active backup is one that can be restored and rolled forward using the current logs to reach the current state of the

database. An ″inactive″ backup cannot be restored and rolled forward to reach the current state of the database because it requires a different set of log files.

Each of the examples shown in the artwork that follows makes the assumption that *num_db_backups* has been set to four.

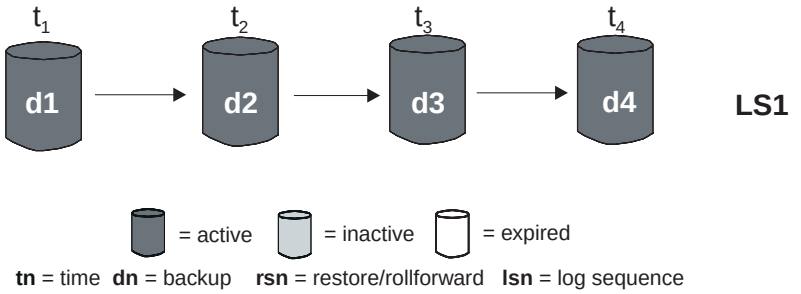


Figure 39. Active Database Backups

All database backups that are no longer needed are marked as “expired”. These backups are considered no longer needed because there are several database backups as defined by *num_db_backups* that are more recent. All table space backups and load backup copies that were taken before the database backup expired are also marked as “expired”.

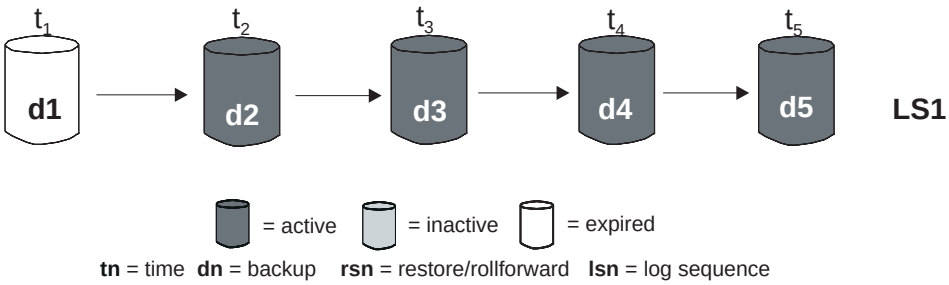


Figure 40. Expired Database Backups

All database backups that are marked as “inactive” and were taken previous to the point in time when the expired database backup was taken are also marked as “expired”. All associated “inactive” table space backups and load backup copies are also marked as “expired”.

When DATALINK columns are involved with the backups as described in the next section, all Data Links servers running the DB2 Data Links Manager are contacted to request the garbage collection of the associated Data Links server files unlinked before the backup that has expired. After physically deleting these backups based on the information contained in the history file, you can

use the PRUNE HISTORY command to remove “expired” entries from the history file. If you do not explicitly prune the history file, the next database backup will cause the “expired” entries to be pruned automatically.

DB2 Garbage Collection also is responsible for marking the history file entries for a DB2 database or table space backup as “inactive” if that backup does not correspond to the current *log sequence* also called the current *log chain*. The current log sequence is determined by the DB2 database backup that has been restored and the log files that have been processed. Once a database backup is restored, all database backups that were taken after the backup that was restored become “inactive”, since the restored backup begins a new log chain. A table space backup becomes “inactive” when, after restoring it, the current state of the DB2 database cannot be reached by applying the current sequence of logs. (When a DB2 database or table space backup becomes “inactive”, DB2 Garbage Collection notifies all Data Links servers running DB2 Data Links Manager so that the corresponding set of file backups would also be marked as “inactive”.)

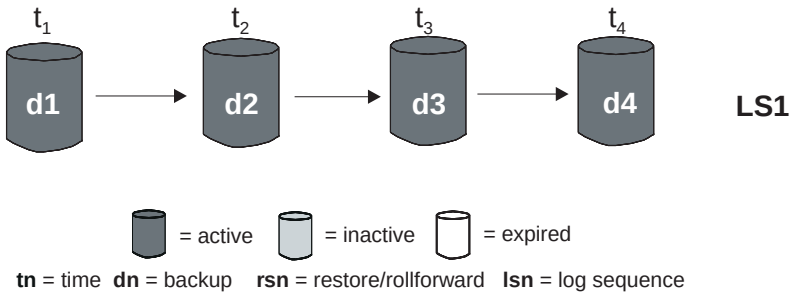


Figure 41. Active Database Backups

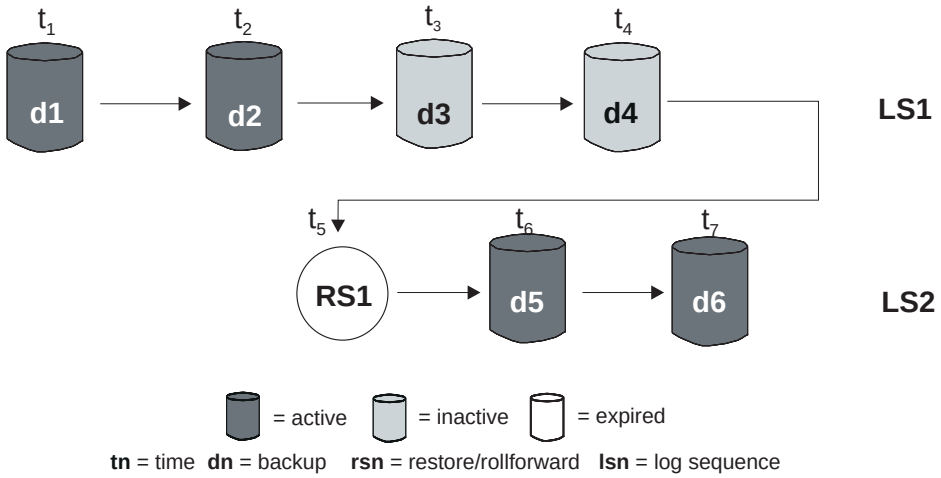


Figure 42. Active Database Backups

DB2 Garbage Collection is invoked after completing a DB2 database backup. The value of the *db2_num_backups* configuration parameter is used to scan the current history file starting with the last entry.

DB2 Garbage Collection is also invoked after completing a restore of a database backup with or without rolling forward the logs.

If an “active” database backup is restored, but it is not the most recent database backup recorded in the history file, any subsequent database backups belonging to the same log sequence are marked as “inactive”.

If an “inactive” database backup is restored, any inactive database backups belonging to the current log sequence are marked as “active” again. As with restoring an “active” database backup, all active database backups that are no longer in the current log sequence are marked as “inactive”.

If DATALINK columns are part of the database backups being performed, DB2 Garbage Collection then contacts all Data Links servers running the DB2 Data Links Manager to make the same status changes to the corresponding set of file backups on the Data Links servers.

After every full database backup, the *rec_his_retentn* configuration parameter is used to prune “expired” entries from the history file. All “expired” backups are removed.

The PRUNE HISTORY command can be used by you at any time to prune only backups marked as “expired” from the history file unless the WITH FORCE option is used. (If a backup is pruned that is not “expired”, all Data

Links servers are contacted and requested to flag for garbage collection the corresponding set of file backups.)

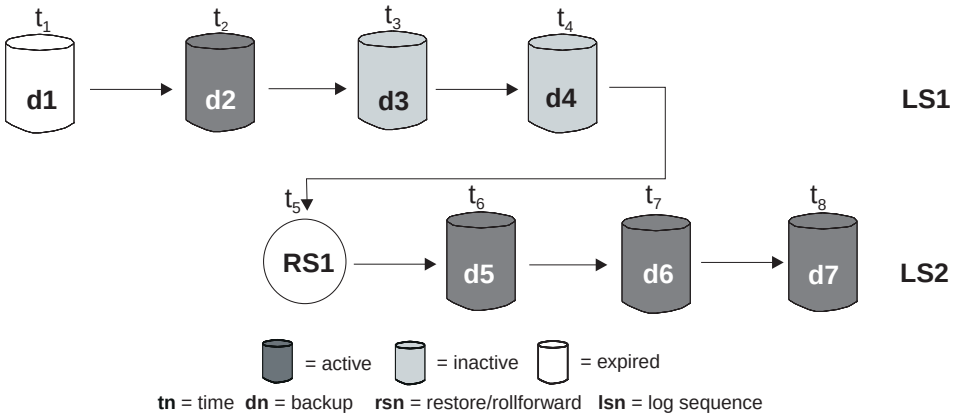


Figure 43. Mixed Active, Inactive, and Expired Database Backups

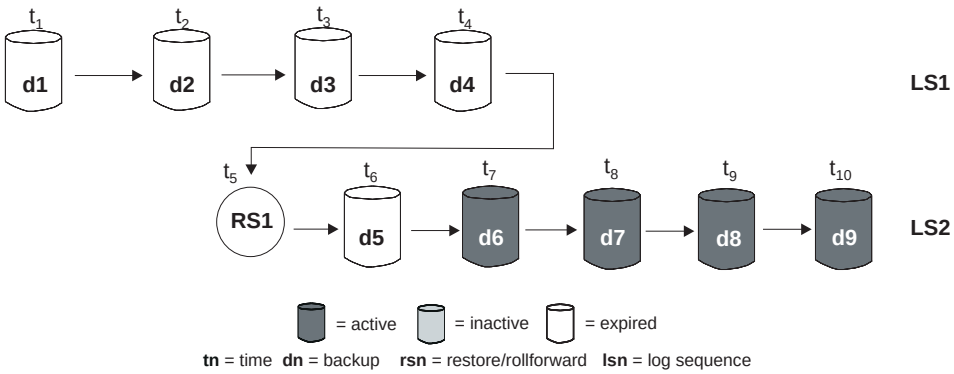


Figure 44. Expired Log Sequence

DB2 Data Links Manager Considerations

The following sections provide information that applies if you have tables that contain DATALINK columns. For a full description of DATALINK columns, refer to the CREATE TABLE statement in the *SQL Reference*.

Crash Recovery Considerations

When an application issues SQL requests involving Data Links servers running the DB2 Data Links Manager (using DATALINK columns with the FILE LINK CONTROL attribute), the database manager distributes the work to the Data Links servers. It also keeps track of which Data Links servers are

involved in the transaction. When the application issues a COMMIT for a transaction, the database manager commits the transaction by using two-phase commit protocol. In the first phase, the database manager writes a PREPARE log record and distributes a PREPARE request to all Data Links servers. Each Data Links server then responds with one of the following:

- YES; signifying that the Data Links server is prepared to commit
- NO; because of an error, Data Links server is not prepared to commit.

The first phase is considered successful if all Data Links servers respond "YES".

The processing in the second phase depends on the outcome of the first phase. If at least one of the Data Links servers responded "NO", the database manager distributes ABORT requests to all the Data Links servers involved. The transaction is rolled back and the error message SQL0903N with reason code "03" is returned to the application. Otherwise, the database manager proceeds to commit the transaction as it usually does in the absence of involvement of Data Links servers. At the end of this processing, it distributes a COMMIT request to all the Data Links servers involved in the transaction.

If a failure occurs on a Data Links server leaving some transactions in the PREPARED state, these transactions are called *indoubt transactions*. The database manager is responsible for tracking the outcome of these transactions and eventually resolving them on the Data Links server. Whenever the database manager determines that a failure has potentially created indoubt transactions on a Data Links server, it marks the state of the Data Links server as needing crash recovery. It disallows any SQL requests involving the Data Links server while it is in this state. SQL0357N with reason code "03" is returned to the application which made the SQL request.

At the time of RESTART, ACTIVATE DATABASE, or first CONNECT processing, the database manager attempts to connect to each configured Data Links server and attempts to resolve its indoubt transactions by aborting or committing them. A Data Links server's state is marked as available if all of its indoubt transactions are resolved except those transactions that are also indoubt on the database manager. In the available state, SQL requests involving the Data Links server are allowed. At the end of this attempt to resolve indoubt transactions if the database manager determines that a Data Links server still potentially has indoubt transactions which need resolution, it marks the state of the Data Links server as needing crash recovery. This can happen, for instance, if a Data Links server is not available during RESTART, ACTIVATE DATABASE, or first CONNECT processing; or, the Data Links server encounters a failure during that processing.

While a Data Links server configured to a database is in a state needing crash recovery, the database manager disallows SQL requests involving that

particular Data Links server. SQL requests involving other data in the database are still allowed. The database manager starts a process which asynchronously attempts to complete crash recovery on each Data Links server requiring recovery. When the process successfully completes the crash recovery, the state of the Data Links server is marked as available thereby allowing further SQL requests involving it.

Backup Utility Considerations

DB2 ensures that by the time the backup utility completes, linked files at Data Links servers running the DB2 Data Links Manager are also backed up. (The backup utility can run either online or offline, and the backup image can be of either a database or a table space.) The description that follows only applies to files that are linked by DATALINK columns that have the RECOVERY parameter set to YES. (Files that are referenced by DATALINK columns for which RECOVERY=NO is specified are not backed up.)

When files are linked, the Data Links servers schedule them to be copied asynchronously to an archive server such as ADSM, or to disk. When the backup utility runs, DB2 ensures that all files scheduled for copying have been copied. At the beginning of backup processing, DB2 also ensures that all Data Links servers that are specified in the DB2 configuration file are running. If a Data Links server has one or more linked files, it must be available until the backup operation completes. If a Data Links server becomes unavailable before the backup operation completes, the backup operation is declared as incomplete.

When a file is unlinked, it is either deleted or returned to its previous permissions, depending on the value specified for the ON UNLINK parameter. A successful backup operation can cause the Data Links servers to clean up the archived versions of files on the archive server (either disk or ADSM). The *num_db_backups* database configuration parameter specifies the number of DB2 database backups before archived versions of the files (that were unlinked) are removed. Refer to *Administration Guide, Performance* for more information about this configuration parameter.

When unlinked files are removed, the information about the unlinked files is also removed from the Data Links server registration tables.

Restore and Rollforward Utility Considerations

The information that follows applies if you have a DATALINK column (or columns) that is defined with RECOVERY=YES option for a table. If a table has a DATALINK column defined with the RECOVERY=NO option, the table is put in the *Datalink_Reconcile_Pending* state at the end of the restore operation. See “Reconciling Data Links” on page 450 for more information.

During restore operations, tables with DATALINK columns may be put into one of the following states.

- *Datalink_Reconcile_Not_Possible*

  When a table is in the *Datalink_Reconcile_Not_Possible* state, it is available for unrestricted manipulative actions for columns other than the DATALINK columns. When a DATALINK column is involved in a SELECT statement, a warning is issued. You can issue UPDATE calls to DATALINK columns (with some restrictions: see "Removing a Table from the Datalink_Reconcile_Not_Possible State" on page 449 for details). You cannot issue INSERT and DELETE statements because they involve the DATALINK column.

- *Datalink_Reconcile_Pending*

  When a table is in the *Datalink_Reconcile_Pending* state, it is available for unrestricted manipulative actions for columns other than the DATALINK columns. When a DATALINK column is involved in a SELECT statement, a warning is issued. You cannot issue any DML statements such as UPDATE, INSERT, or DELETE.

These states are reported in the db2diag.log file when the restore or rollforward utilities run. You can also use the **db2dart** command to obtain this information.

When you restore a database or table space and do *not* specify the WITHOUT DATALINK option, the following conditions must be satisfied for the restore operation to succeed:

- All Data Links servers containing the DATALINK data must all be available.
- All Data Links servers that are recorded in the backup file must be available.
- Information about all DATALINK columns that are recorded in the backup file must exist in the appropriate Data Links servers' registration tables.

  If all the information about the DATALINK columns is not recorded in the registration tables, the table with the missing DATALINK column information is put into the *Datalink_Reconcile_Not_Possible* state after the restore operation (or the roll-forward operation, if used) completes.

  If the backup is not recorded in the registration tables, it means that the backup file that is provided is earlier that the value for *num_db_backups* and has already been "garbage collected". This means that the archived files from this earlier backup have been removed and cannot be restored. All tables that have DATALINK columns are put into the *Datalink_Not_Possible* state.

The table remains available to users, but the values in the DATALINK columns may not reference the files accurately (for example, a file may not be found that matches a value for the DATALINK column).

If you do not want this behavior, you can put the table into the *check pending* state by issuing the SET CONSTRAINTS for *tablename* TO DATALINK RECONCILE PENDING command.

If, after a restore operation, you have a table in the *Datalink_Reconcile_Not_Possible* state, you can fix the DATALINK column data in one of the ways suggested under "Removing a Table from the Datalink_Reconcile_Not_Possible State" on page 449.

Note: In the process of marking a file from the unlinked state to the linked state, that file may have to be retrieved from an archive server to the file system. If an error occurs during this process (for example, a file cannot be copied into the file system because of duplicate file names), the corresponding table is placed into the *Datalink_Reconcile_Pending* state.

When you restore a database or table space and you *do* specify the WITHOUT DATALINK option, and one or more of the Data Links servers containing the DATALINK data is not available, all table spaces containing tables with DATALINK values on the unavailable server(s) are placed in the RESTORE PENDING state.

Restoring Databases from an offline Backup without Rolling Forward

Note: You can only restore without rolling forward at the database level, and not the table-space level. To restore a database without rolling forward, you could either restore a nonrecoverable database (that is, a database that uses circular logging), or you would specify the WITHOUT ROLLING FORWARD parameter for the restore utility.

If you use the restore utility with the WITHOUT DATALINK option, all tables with DATALINK columns are placed in the *Datalink_Reconcile_Pending* state and no reconciliation is performed with the Data Links servers during the restore operation.

If you do not use the WITHOUT DATALINK option, and all the Data Links servers are available and all information about the DATALINK columns is fully recorded in the registration tables, the following occurs for each Data Links server recorded in the backup file:

- All files that were linked after the backup image that was used for the database restore are marked as unlinked (because they are not recorded in the backup image as being linked).

- All files that were unlinked after the backup image, but were linked before the backup image was taken, are marked as linked (because they are recorded in the backup image as being linked). If the file was subsequently linked to another table in another database, the restored table is put into the *Datalink_Reconcile_Pending* state.

Restoring Databases and Table Spaces and Rolling Forward to the End of the Logs

If you restore then roll forward the database or table space to the end of the logs (meaning that all logs are provided), a reconciliation check is not required (regardless of whether the WITHOUT DATALINK parameter is specified). If you are not sure whether all the logs were provided for the roll-forward operation, or think that you may need to reconcile DATALINK values:

1. Issue the SQL statement for the table (or tables) involved:

   ```
   SET CONSTRAINTS FOR tablename TO DATALINK RECONCILE PENDING
   ```

   This puts the table in *Datalink_Reconcile_Pending* state and *check-pending* state.
2. If you do not want a table in the *check-pending* state, issue the following SQL statement:

   ```
   SET CONSTRAINTS FOR tablename IMMEDIATE CHECKED
   ```

   This takes the table out of the *check-pending* state, but leaves it in the *Datalink_Reconcile_Pending* state. You must use the reconcile utility to take the table out of this state. For more information, see “Reconciling Data Links” on page 450.

Restoring Databases and Table Spaces and Rolling Forward to a Point in Time

When working with Data Links tables, you can roll-forward to the end of the logs or to a specified point-in-time.

Tables in table spaces that are rolled forward to a point-in-time are placed in the *Datalink_Reconcile_Pending* state at the end of the roll-forward operation. You should use the reconcile utility to remove them from this state. For more information, see “Reconciling Data Links” on page 450.

Point-in-Time Roll-Forward Example

Following is a simple scenario showing the files that need to be retained in order to handle backup and recovery. The example shows changes to the value of a single row in column of type DATALINK together with the files

that the DB2 Data Links Manager needs to retain to support recovery. For this example, the assumption is made that there is no support for point-in-time recovery of these files earlier than the last backup. Data Links servers running the DB2 Data Links Manager do not have such a restriction. Observe that fileA exists until time 3, at which time it is deleted because it was unlinked at time 2, and the policy for the database in this example is to keep the unlinked files until the next backup is run (that is, the *num_db_backups* database configuration parameter is set to 1).

Time	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Activity	Create	Update	Backup	Update	Update	Delete	Restore to 5
Column Value	valueA	valueB	valueB	valueC	valueD	-	valueD
Linked File	fileA	fileB	fileB	fileC	fileD	-	fileD
Extra Files Kept by Data Links File Manager		fileA		fileB	fileB, fileC	fileB, fileC, fileD	fileB, fileC

Note: Recovery of linked files is always done in conjunction with the rest of the database.

DB2 Data Links Manager and Recovery Interactions

The following table shows the different types of recovery that you can perform, the DB2 Data Links Manager processing that occurs during restore and roll-forward processing, and whether you need to run the Reconcile utility after the recovery is complete:

Type of Recovery	**DB2 Data Links Manager Processing during Restore**	**DB2 Data Links Manager Processing during Rollforward**	**Reconcile**
Non-recoverable database (*logretain*=NO)			
Database restore	Fast reconcile is performed	N/A	Can be optionally run if problem with file links is suspected
Database restore using WITHOUT DATALINK option	Tables put in *Datalink_Reconcile_Pending* state	N/A	Required
Recoverable database (*logretain*=YES)			

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Database restore using WITHOUT ROLLING FORWARD option	Fast reconcile is performed	N / A	Optional
Database restore using WITHOUT ROLLING FORWARD and WITHOUT DATALINK options	Tables put in *Datalink_Reconcile _Pending* state	N / A	Required
Database restore and roll forward to end of logs	No action	No action	Optional
Database restore using WITHOUT DATALINK option and roll forward to end of logs	No action	No action	Optional
Table space restore and roll forward to end of logs	No action	No action	Optional
Table space restore using WITHOUT DATALINK option and roll forward to end of logs	No action	No action	Optional
Database restore and roll forward to a point in time	No action	Tables put in *Datalink_Reconcile _Pending* state	Required
Database restore using WITHOUT DATALINK option and roll forward to a point in time	No action	Tables put in *Datalink_Reconcile _Pending* state	Required
Table space restore and roll forward to a point in time	No action	Tables put in *Datalink_Reconcile _Pending* state	Required
Table space restore using WITHOUT DATALINK option and roll forward to a point in time	No action	Tables put in *Datalink_Reconcile _Pending* state	Required

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Database restore to a different database name, alias, hostname, or instance with no roll forward (see note 449)	Tables put in *Datalink_Reconcile _Not_Possible* state	N/A	Optional, but tables in *Datalink_Reconcile _Not_Possible* state must be manually fixed
Database restore to a different database name, alias, hostname or instance and roll forward	No action	Tables put in *Datalink_Reconcile _Not_Possible* state	Optional, but tables in *Datalink_Reconcile _Not_Possible* state must be manually fixed
Database restore from an unusable backup (image has been garbage-collected on the Data Links server) with no roll forward (see note 449)	Tables put in *Datalink_Reconcile _Not_Possible* state	N/A	Optional, but tables in *Datalink_Reconcile _Not_Possible* state must be manually fixed
Database restore from an unusable backup (image has been garbage-collected on the Data Links server) and roll forward	No action	Tables put in *Datalink_Reconcile _Not_Possible* state	Optional, but tables in *Datalink_Reconcile _Not_Possible* state must be manually fixed
Table space restore from an unusable backup (image has been garbage-collected on the Data Links server) and roll forward	No action	Tables put in *Datalink_Reconcile _Not_Possible* state	Optional, but tables in *Datalink_Reconcile _Not_Possible* state must be manually fixed
Database restore from an unusable backup (image has been garbage-collected on the Data Links server) using the WITHOUT DATALINK option and no roll forward (see note 449)	Tables put in *Datalink_Reconcile _Pending* state	N/A	Required

Type of Recovery	DB2 Data Links Manager Processing during Restore	DB2 Data Links Manager Processing during Rollforward	Reconcile
Database restore from an unusable backup (image has been garbage-collected on the Data Links server) using the WITHOUT DATALINK option and roll forward	No action	Tables put in *Datalink_Reconcile _Not_Possible* state	Optional, but tables in *Datalink_Reconcile _Not_Possible* state must be manually fixed
Table space restore from an unusable backup (image has been garbage-collected on the Data Links server) using the WITHOUT DATALINK option and roll forward	No action	Tables put in *Datalink_Reconcile _Not_Possible* state	Optional, but tables in *Datalink_Reconcile _Not_Possible* state must be manually fixed

Note:

A restore using an offline backup and the WITHOUT ROLLING FORWARD option (*logretain* is on), or a restore using an offline backup (*logretain* is off).

Removing a Table from the Datalink_Reconcile_Not_Possible State

A restored table (or tables) with a DATALINK column is put into the *Datalink_Reconcile_Not_Possible* state:

- If a table space is restored from a backup that is earlier than the value specified for the *num_db_backups* database configuration parameter. Refer to *Administration Guide, Performance* for more information on this configuration parameter.

DB2 still allows the table to be accessed, even though the DATALINK column values may not be valid. If you want to prevent access to a table with possibly inconsistent DATALINK column values, issue the SET CONSTRAINTS for *tablename* TO DATALINK RECONCILE PENDING command. You can update the DATALINK values as follows:

- Using the SQL UPDATE statement, set the data location part of a DATALINK column value to a zero-length URL if the column is not nullable, or to NULL if the column is nullable.
- Restore the files on the appropriate Data Links servers. Then run an application that issues SELECT statements to read the DATALINK column

values, and issues UPDATE statements to update the DATALINK column with the same values. Note that the *Datalink_Reconcile_Not_Possible* state must be on while the DATALINK column values are being updated. After the update operation completes, the files will be marked as linked on the appropriate Data Links servers.

You then reset the *Datalink_Reconcile_Not_Possible* state by issuing the following command:

```
SET CONSTRAINTS FOR tablename DATALINK RECONCILE PENDING IMMEDIATE UNCHECKED
```

Reconciling Data Links

You use the reconcile utility to reconcile data links. The utility is initiated from DB2, and involves all the Data Links servers running the DB2 Data Links Manager that are referenced by the DATALINK column values. It validates that the referenced files either exist on the Data Links server, or that links can be re-established. The following sections describe how DB2 detects whether you need to reconcile data links, and how to reconcile them.

If a Data Links server file reference does not exist or cannot be re-established, the reconcile utility places a copy of the rows in error along with a reason for each into an exception table (if specified), then modifies the offending rows. If the exception table is not specified, the DATALINK column values for which a file reference could not be re-established are copied to an exception report file along with a column-ID and reason. You can use the exception table (if specified) information or the report to update the rows to make the required corrections. The exception table used with the reconcile utility is identical to the exception table used by the Load utility. Refer to *Data Movement Utilities Guide and Reference* for more information on the Load utility. The report uses the naming convention *report*.exp (the .exp extension is supplied by the reconcile utility). For example, you can invoke the reconcile utility with the following statement:

```
db2 RECONCILE dept DLREPORT  /u/scottba/report FOR EXCEPTION excptab
```

This command reconciles the table called dept, and writes exceptions to the exception table excptab, which was created by the user. Information about files that were unlinked during reconciliation are written to the file report.ulk, which is created in the directory /u/scottba. If *FOR EXCEPTION excptab* is not specified, then the exception information is written to the file report.exp, which is created in the directory /u/scottba. Refer to the *Command Reference* for more information on the reconcile utility.

Detection of Situations That Require Reconciliation

Following are some situations when you may need to run the reconcile utility:

- The entire database is restored and rolled forward to a point in time. Because the *entire* database is rolled forward to a committed transaction, no tables will be in the *check pending* state (due to referential constraints or check constraints). All data in the database is brought to a consistent state. The DATALINK columns, however, may not be synchronized with the metadata in the DB2 Data Links Manager, and reconciliation is required.

  In this situation, tables with DATALINK columns data will already be in the *Datalink_Reconcile_Pending* state. You should issue the reconcile utility for each of these tables.
- A particular Data Links server running the DB2 Data Links Manager loses track of its metadata. This can occur for different reasons. For example:
  - The Data Links server was cold started.
  - The Data Links server metadata was restored to a back-level state.

  In some situations, such as SQL UPDATEs and DELETEs, DB2 may be able to detect a problem with the metadata in a Data Links server. In these situations, DB2 would fail the SQL statement. You would put the table in the *Datalink_Reconcile_Pending* state by using the SET CONSTRAINTS statement, then run the reconcile utility on that table.
- A file system is not available (for example, because of a disk crash) and is not restored to the current state. In this situation, files may be missing.

  An error like this will typically be discovered by an application when it cannot access the file whose file reference it obtained from the database. You should put the table in the *Datalink_Reconcile_Pending* state and run the reconcile utility on it. Some of the files may be restored from the archive server if their corresponding DATALINK columns had RECOVERY=YES. In any case, the reconcile utility will record the exceptions in the exception table or in the exception report. You can then restore those files or issue SQL UPDATEs to fix the column.

Summary of Procedure for Reconciliation

If you need to reconcile data links because of point-in-time recovery or because Data Links servers running the DB2 Data Links Manager and DB2 control information do not match:

1. Put the table in the *Datalink_Reconcile_Pending* state by issuing the SET CONSTRAINTS statement. (In some situations, DB2 will do this for you.)
2. Use the reconcile utility to resolve the links, and take the appropriate actions for the exceptions in the exception table or in the exception report.

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. You can use ADSM Client Version 3.1.x.3 and later with DB2. 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. On SunOS and Solaris environments, perform the following steps. (For other UNIX-based platforms, begin at step 2.)
   a. Ensure that the required level of operating system is installed: SunOS 5.5.1 or Solaris 2.5.1.
   b. Install the ADSM Client Version 3.1.x.3 or later. Ensure that you remove *all* previous ADSM packages before installing this version of the client.
   c. Verify that ADSM is installed in the directories `/opt/IBMDSMap5`, `/opt/IBMDSMba5`, and `/opt/IBMDSMsa5`.
   d. Create the following symbolic links in the directory `/usr/lib`, if they do not already exist:

      ```
      libApiDS.so -> libApiDS.so.1
      libApiDS.so.1 -> /opt/IBMDSMap5/api/libApiDS.so.2
      ```
2. Create or modify the ADSM user configuration options file `/usr/sbin/dsm.opt`, and the ADSM system configuration options file `/usr/sbin/dsm.sys` to suit your environment.
3. On SunOS and Solaris environments, perform the following steps. (For other UNIX-based platforms, continue at step 4.)
   a. Copy `/usr/sbin/dsm.opt` and `/usr/sbin/dsm.sys` to the directory `/opt/IBMDSMap5`.
   b. Copy `/opt/IBMDSMap5/solaris/dsmaptica` to the directory `/opt/IBMDSMap5`.
4. Set the environment variables used by ADSM:

DSMI_DIR identifies the user-defined directory path where the API trusted agent file (`dsmapicta` or `dsmtca`) are located.

Note: For the SunOS and Solaris environments, this should be set to `/opt/IBMDSMap5`.

DSMI_CONFIG
identifies the user-defined directory path to the `dsm.opt` file, which contains the ADSM user options.

Note: For the SunOS and Solaris environments, this should be set to `/opt/IBMDSMap5/dsm.opt`.

DSMI_LOG identifies the user-defined directory path where the error log (`dsierror.log`) will be created.

5. 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.

6. 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* or *dsmtca*) 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 OS/2 and Windows NT 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 6 000 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.
- On non-UNIX-based platforms, the backup and restore utilities do not allow more than one (1) ADSM session.

The current ADSM client on the Windows operating system and OS/2 is non-reentrant, and so multiple sessions cannot be created with the backup, restore, or load utilities from a single machine.

In a single node configuration, if a user attempts to issue a backup command such as:

```
db2 backup db sample use adsm open 3 sessions
```

DB2 will detect that multiple sessions are not supported by ADSM, and will return SQL2032N. The equivalent scenario also applies to load copies using ADSM.

However, in a multiple logical node (MLN) configuration on Windows NT, DB2 may not be able to detect the use of multiple sessions on a single machine if each logical node attempts to create only one session. If multiple logical nodes are being backed up, restored, or loaded in parallel using ADSM, DB2 will allow the operation to proceed if each node attempts to use a single session, even though the logical nodes actually reside on the same physical hardware. This can lead to failed backup attempts, and hung load processes, and should not be attempted.

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:

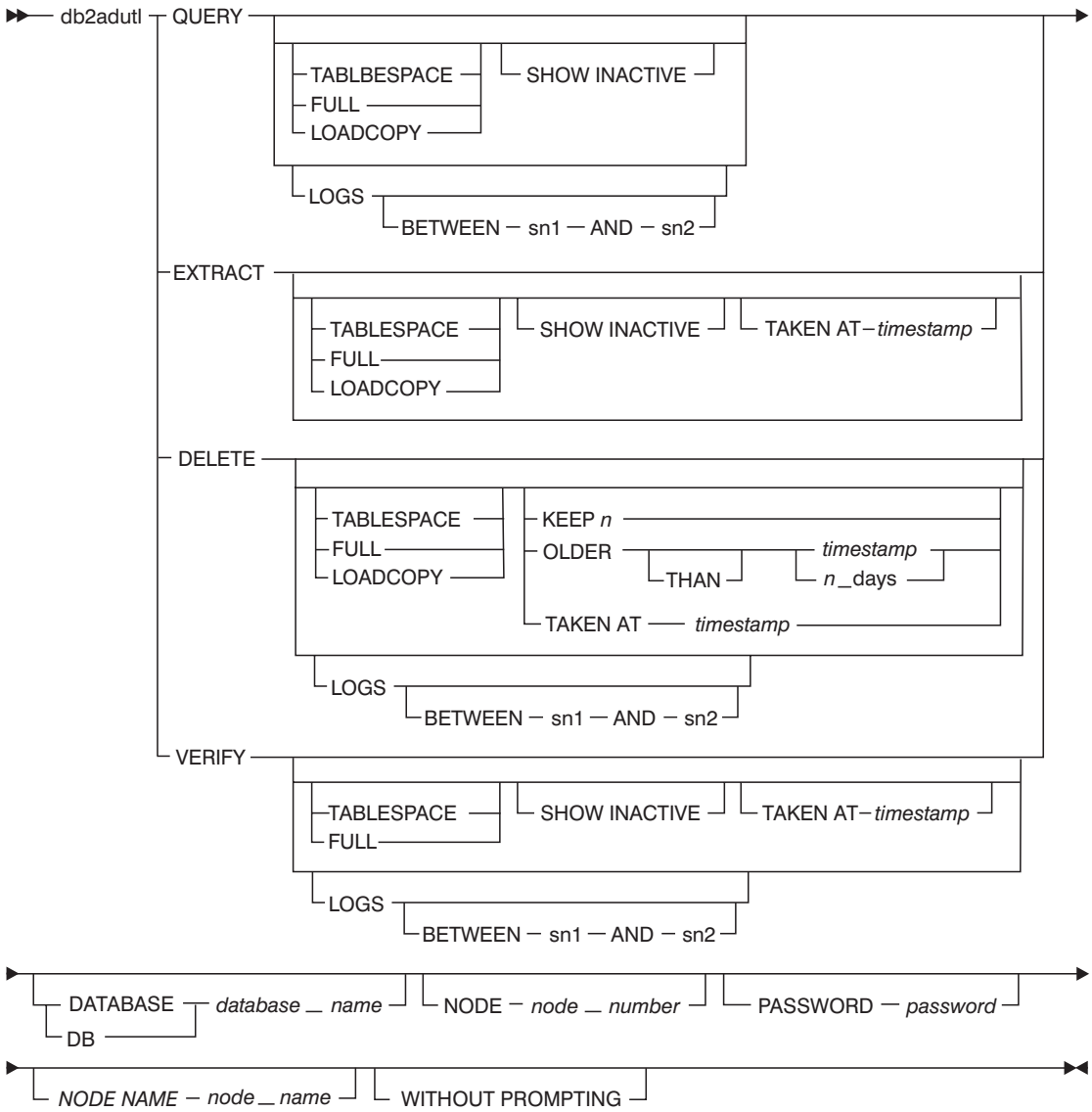


Figure 45. Syntax for db2adutl

Where:

QUERY
Queries the ADSM server for DB2 objects.

EXTRACT
Copies DB2 objects from the ADSM server to the local machine and directory.

DELETE
Either deactivates backup objects or deletes log archives on the ADSM server.

VERIFY
Performs consistency checking on the backup copy that is on the server. (Note that this parameter causes the entire backup image to be transferred over the network.)

TABLESPACE
Includes only table space backup images.

FULL Includes only full database backup images.

LOADCOPY
Includes only load copy images.

LOGS Includes only log archive images.

BETWEEN *sn1* **AND** *sn2*
Specifies to use the logs between log sequence number 1 and log sequence number 2.

SHOW INACTIVE
Includes backup objects that have been deactivated.

TAKEN AT *timestamp*
Specifies a backup image by its timestamp.

KEEP *n*
Deactivates all objects of the specified type except for the most recent *n* by timestamp.

OLDER THAN *timestamp* **or** *n_days*
Specifies that objects with a timestamp earlier than *timestamp* or *n* days will be deactivated.

DATABASE *database_name*
Specifies to work with objects associated with *database_name* only.

NODE *node_number*
Specifies to work with objects created by node *node_number* only.

PASSWORD *password*
Specifies the ADSM client password for this node (if required). If a specific database is specified and the password is not provided, the value specified for the *adsm_password* database configuration parameter is passed to ADSM; otherwise, no password is used.

NODENAME *node_name*
Specifies to work with images associated with a specific ADSM node name only.

WITHOUT PROMPTING
You are not prompted for verification before objects are deleted.

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 table space backup information.
   table space backup image: 1, Time: 19970929094003,
                               Oldest log: S0000051.LOG, Sessions used: 1
   table space backup image: 2, Time: 19970929093043,
                               Oldest log: S0000050.LOG, Sessions used: 1
   table space 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 table space backup information.
   table space backup image: 1, Time: 19950929094003,
                           Oldest log: S0000051.LOG, Sessions used: 1
   table space backup image: 2, Time: 19950929093043,
                           Oldest log: S0000050.LOG, Sessions used: 1
   table space 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 3. Distributed Transaction Processing

Chapter 10. 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 (such as Windows platforms), 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:

- "Using a Single Database in a Transaction" on page 466
- "Using Multiple Databases in a Single Transaction" on page 467

For information on creating applications using distributed databases, refer to the *Application Development 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*.

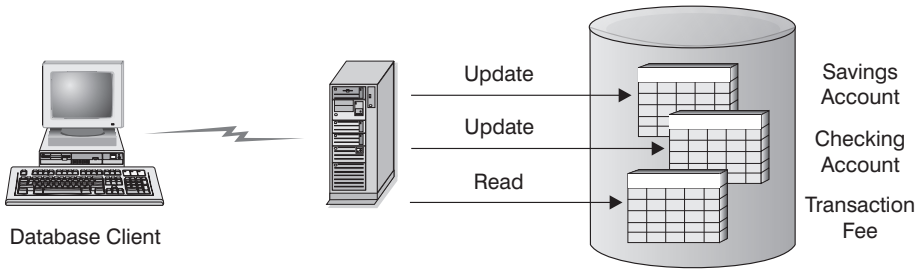


Figure 46. Using a Single Database in a Transaction

Figure 46 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 4. Implementing Your Design" on page 99)
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 *Application Development 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 469.

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 *multisite update* or *two-phase commit*. See "Updating Multiple Databases" on page 469 for another example of a multisite update.

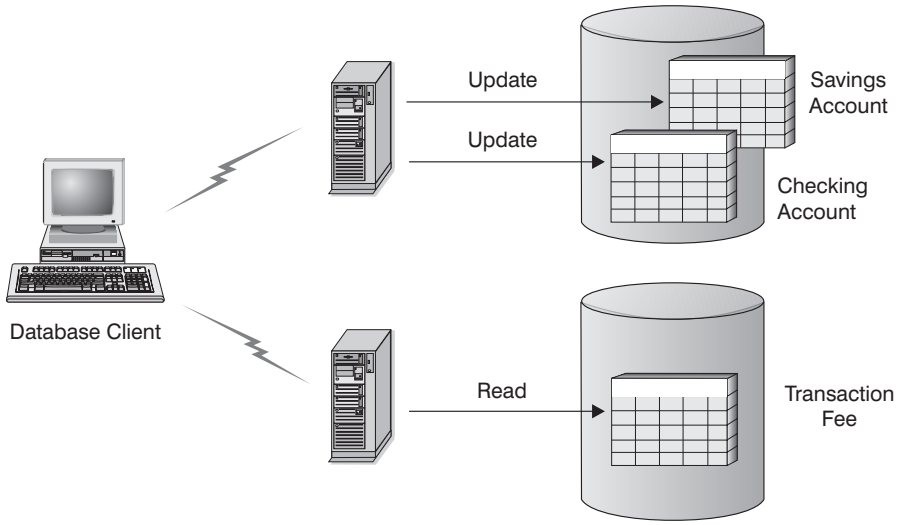


Figure 47. Using Multiple Databases in a Single Transaction

Figure 47 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 466, 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 4. Implementing Your Design" on page 99)
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 *Application Development 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.

Host and AS/400 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. Refer to the *DB2 Connect User's Guide* for additional information on how to set up and use DB2 Connect.
- Refer to the *DRDA Connectivity Guide* for more information on connectivity issues.

Federated Server and Pass-Through Session Additional Information:

- You cannot perform insert, update, or delete operations against nicknames.
- If you open a pass-through session to a data source, DB2 assumes that you intend to insert, update, or delete information at that data source. Therefore, if your application updates information in one database and then opens a pass-through session to a data source, within the same unit-of-work, you must specify SYNCPOINT(TWOPHASE) on the PREP command.
- Refer to the *SQL Reference* for more information on pass-through sessions (SET PASSTHRU).

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 a *multisite update*. This type of environment is more complex than that described in "Updating a Single Database" on page 467. As a result, additional topics will be introduced below.

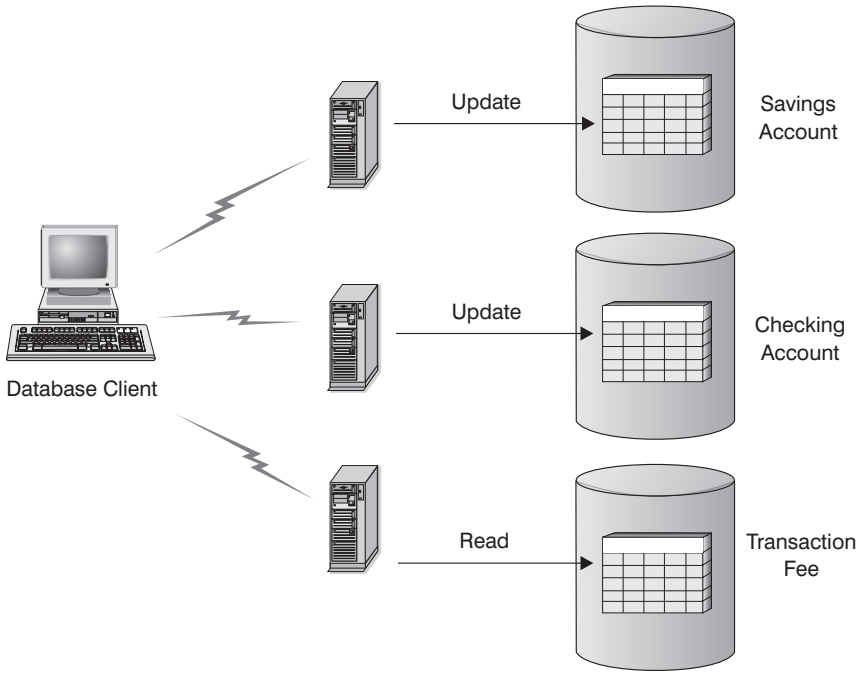


Figure 48. Updating Multiple Databases

Figure 48 shows an example similar to Figure 47 on page 468, 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 467.

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 4. Implementing Your Design" on page 99)
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 *Application Development 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 the updating of 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.

If you are using an XA-compliant transaction manager such as IBM TXSeries, BEA Tuxedo, or Microsoft Transaction Series, please see "Chapter 11. Using DB2 with an XA-Compliant Transaction Manager" on page 489 for integration instructions.

When using DB2 UDB to coordinate your transactions you need to meet certain configuration requirements. If you use TCP/IP exclusively for communications and DB2 UDB and DB2 for OS/390 are the only database server involved in your transactions then configuration is simplified over environments that do not meet these criteria.

DB2 UDB and DB2 for OS/390 Using TCP/IP Connectivity: If all the following are true in your environment:

- All communications with remote database servers uses TCP/IP exclusively
- DB2 Universal Database or DB2 for OS/390 are the only database servers involved in the transaction
- DB2 for OS/390 access is *not* via the DB2 Syncpoint Manager.

  The DB2 Syncpoint Manager is required when:
  - SNA connectivity is used with host or AS/400 database servers for multi-site updates.
  - An XA-compliant transaction manager (such as IBM TXSeries CICS) is coordinating the two-phase commit.

This applies to both SNA and TCP/IP connectivity with host or AS/400 database servers. For detailed information, see "Chapter 11. Using DB2 with an XA-Compliant Transaction Manager" on page 489.

then the configuration steps for multisite update are simplified. There is no need to catalog the Transaction Manager Database at each remote database server. Nor is there a need to catalog each remote database server at the Transaction Manager database instance. This information is exchanged between the DB2 client, the designated transaction manager database instance, and the DB2 UDB and/or DB2 for OS/390 database servers automatically without manual database configuration.

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*. Refer to "Configuring DB2" in *Administration Guide, Performance* for more information on this configuration parameter. Consider the following factors when setting this configuration parameter:

- The transaction manager database can be:
  - A DB2 UDB database
  - A DB2 for OS/390 Version 5 or later database.

    This is the recommended database server to use as the transaction manager database. OS/390 systems are, generally, more secure than workstation servers. This reduces the possibility of accidental power downs, reboots, and so on. Therefore the recovery logs, used in the event of resynchronization, are more secure.
- 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.

  Care must be taken when using `1ST_CONN`. You should only use this configuration if it is easy to ensure that all involved databases are cataloged correctly, 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.

Other Environments: If your transactions involve any of the following situations:

- TCP/IP is not used exclusively for communications with remote database servers (for example, NETBIOS is used)
- DB2 Common Server V2 database server is accessed
- DB2 for MVS V3 or V4, DB2 for AS/400, or DB2 for VM&VSE is accessed
- DB2 for OS/390 is accessed using SNA
- The DB2 Syncpoint Manager is used to access host or AS/400 database servers

then the configuration steps for multisite update are more involved than the preceding discussion.

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*. Refer to "Configuring DB2" in *Administration Guide, Performance* for more information on this configuration parameter. Consider the following factors when setting this configuration parameter:

- The transaction manager database can be:
  - A DB2 Universal Database database

    This is the recommended database.
  - A DB2 common server V2 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 alternative 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 situation, 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. You should only use this configuration if it is easy to ensure that all involved databases are cataloged correctly, 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 481).

Other Configuration Considerations in Any Environment

You should consider the values of the following configuration parameters when you are setting up your environment. For additional information about setting these parameters, also refer to the *DB2 Connect User's Guide*.

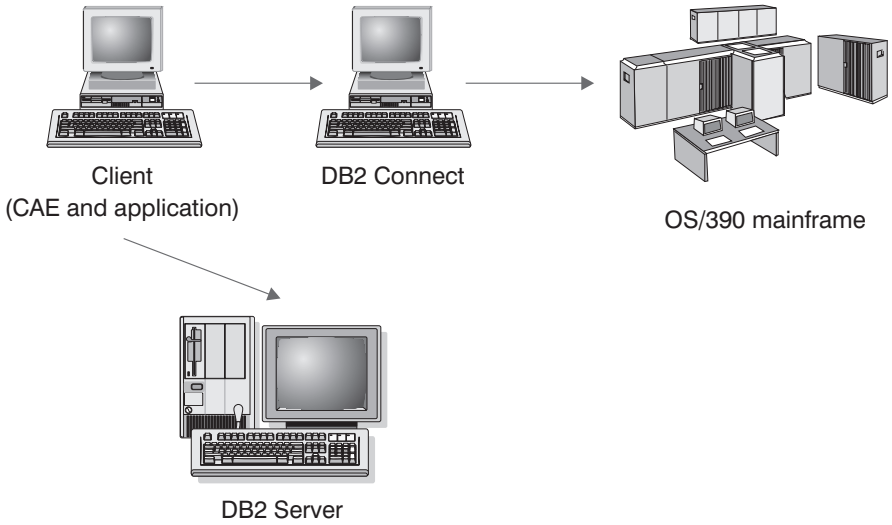


Figure 49. Configuration Considerations

- The following are database manager configuration parameters:
  - *tm_database*

    The *tm_database* database manager configuration parameter identifies the name of the Transaction Manager (TM) database for each DB2 instance.
  - *spm_name*

    The *spm_name* database manager configuration parameter identifies the name of the DB2 Syncpoint Manager (SPM) instance to the database manager. It is defined in the system database directory and, if remote, in the node directory.

    For resynchronization to be successful, the name must be unique across your network.
  - *resyn_interval*

    The *resync_interval* database manager configuration parameter identifies the time interval in seconds for which the DB2 Transaction Manager (TM) database manager, DB2 server database manger, and the DB2 Syncpoint Manager (SPM) should retry the recovery of any outstanding indoubt transactions.
  - *spm_log_file_sz*

    The *spm_log_file_sz* database manager configuration parameter identifies the SPM log file size in 4K pages.
  - *spm_max_resync*

    The *spm_max_resync* database manager configuration parameter identifies the number of agents that can simultaneously perform resync operations.
  - *spm_log_path*

The *spm_log_path* database manager configuration parameter identifies the log path for the SPM log files.

- The following are database configuration parameters:
  - *maxappls*

    The *maxappls* database configuration parameter specifies the maximum number of active applications allowed.

    The value of this parameter must be equal to or greater than the sum of the connected applications plus the number of these same applications which may be concurrently in the process of completing a two-phase commit or rollback. This sum should then have added to it the anticipated number of indoubt transactions which might exist at any one time. See "Recovering from Problems During Two-Phase Commit" on page 481 for more information on indoubt transactions.

    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.

  - *autorestart*

    This database configuration parameter specifies whether the RESTART DATABASE routine will be invoked automatically 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.

Refer to *Administration Guide, Performance* for more information on these configuration parameters.

Note: Before installing either a fixpack for, or a new version of DB2, you should ensure that no indoubt transactions exist with any host or AS/400 server. Use the LIST DRDA INDOUBT TRANSACTION command to determine if any indoubt transactions exist.

Host or AS/400 Applications Which Access a DB2 Universal Database Server in a Multisite Update

In this situation, SNA connectivity is the only type supported. The DB2 Syncpoint Manager is required in order to permit multisite update. DB2 Universal Database does not support multisite update from host or AS/400 database clients using TCP/IP connectivity.

The database server which is being accessed from the host or AS/400 database client does not have to be local to the workstation which has the DB2 Syncpoint Manager. The host or AS/400 database client could connect to a DB2 UDB server using the DB2 Syncpoint Manager workstation as an interim gateway. This allows you to isolate the DB2 Syncpoint Manager workstation in a secure environment while the actual DB2 UDB Servers are remote in your organization. This also permits DB2 Common Server V2 database to be involved in multisite updates originating from host or AS/400 database clients.

The steps are as follows:

- On the workstation which has the DB2 Syncpoint Manager:
  1. Install DB2 Universal Database Enterprise Edition or Enterprise Edition - Extended in order to provide multisite update support with host or AS/400 database clients.
  2. Create a database instance on the same system. For example, you can use the default instance DB2, or use the following command to create a new instance:

     ```
     db2icrt myinstance
     ```
  3. Supply licensing information as required.
  4. Ensure that the registry value db2comm includes the value APPC.
  5. Configure SNA communications as required.

     The configuration will be easier if the SPM_NAME value is the same as the LU name, and the SPM uses the same LU as the DB2 Connect workstation.

     On an AIX system, the SPM_NAME value *must* be same as the names of:

     a. The transaction program (TP) profile used by the SPM.
     b. The local side information profile used by the SPM.
     c. The SPM instance name on application requesters.
  6. Determine the value to be specified for the SPM_NAME database manager configuration parameter, and optionally values for the SPM_LOG_FILE_SZ and SPM_MAX_RESYNC database manager configuration parameters if the defaults are not appropriate for your situation.

7. Update SPM_NAME on the DB2 Universal Database server. For example, you can use the following command:

   ```
   update database manager configuration using spm_name SPMNAME
   ```

8. Configure communications as required for this DB2 workstation to connect to remote DB2 UDB servers, if any.
9. Configure communications as required for remote DB2 UDB Servers to connect to this DB2 Syncpoint Manager workstation.
10. Stop and restart the database manager on the DB2 Universal Database server to start the SPM for the first time.

    You should be able to connect to the remote DB2 UDB servers from this DB2 Syncpoint Manager workstation.

- On Each Remote DB2 UDB Server which will be accessed by the Host or AS/400 Database Client

  The database administrator must also perform the following steps at each remote system where a DB2 UDB Server will be accessed by a host or AS/400 database client.

  The remote DB2 UDB Server must have its communications support configured so that the remote DB2 Syncpoint manager can connect to it AND so that this DB2 UDB Server can connect to the remote DB2 Syncpoint Manager.

  1. Configure communications as required for the remote DB2 Syncpoint Manager (SPM) workstation to connect to this DB2 UDB Server.
  2. Catalog the database directory entry for the remote DB2 Syncpoint Manager instance, and catalog the node entry for the location of the SPM. For example:

     ```
     catalog tcpip node SPMNODE remote SERVERD server db1inst1c
     db2 catalog database SPMNAME as SPMNAME at node SPMNODE
     ```

     The Database alias name *must* be the same as the actual database name.

     You should be able to connect to the SPM from the DB2 Client.
  3. Stop and restart the database manager on the application requester.

Understanding the Two-Phase Commit Process

The following example illustrates the steps of the example transaction (described in “Updating Multiple Databases” on page 469) 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.

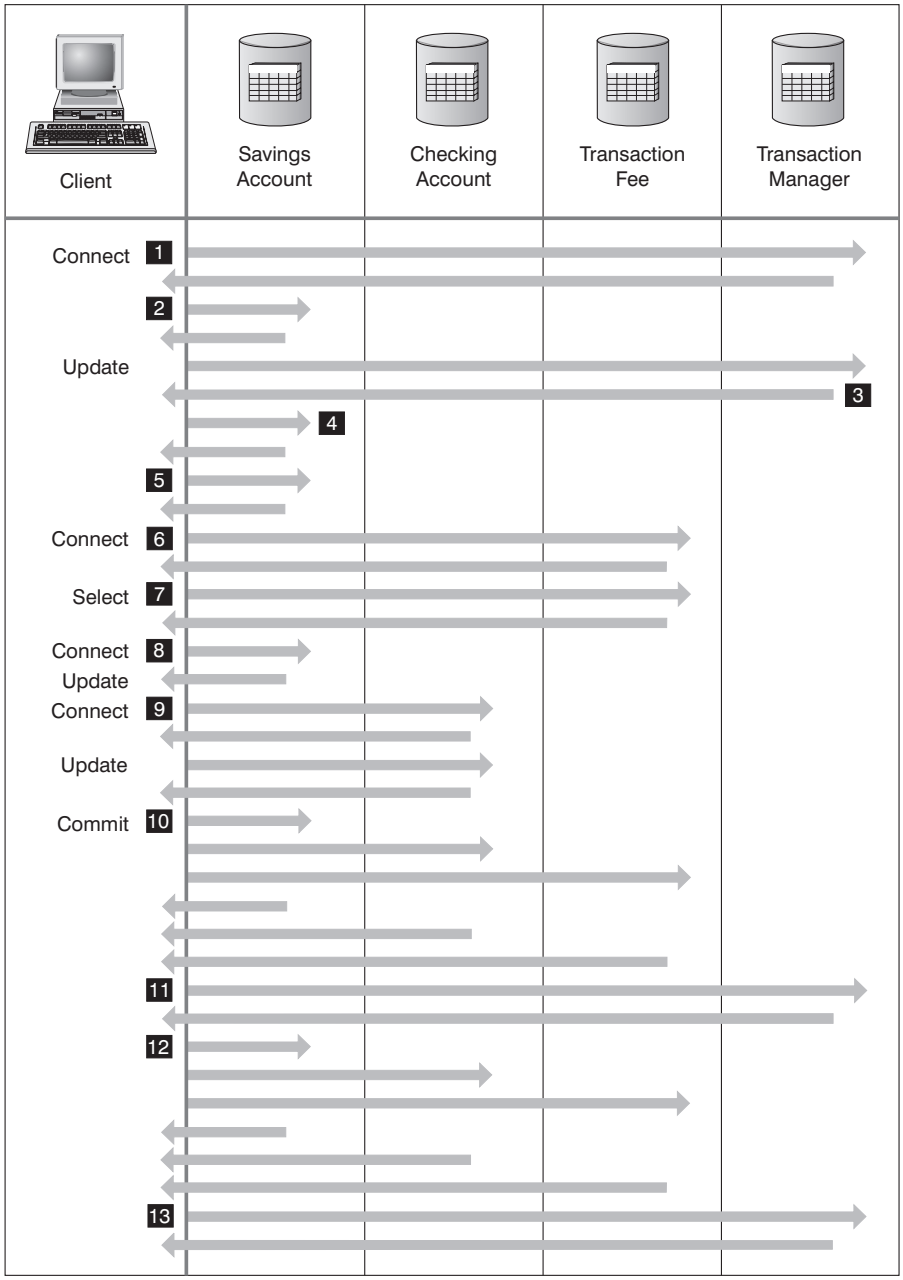


Figure 50. Updating Multiple Databases

0 The application is prepared for two-phase commit. This can be accomplished through precompilation options (refer to the *Application*

Development Guide for details). This can also be accomplished through the DB2 CLI configuration (refer to the *CLI Guide and Reference* for details).

1 When the database client wants to connect to SAVINGS_DB, it first internally connects to the Transaction Manager (TM) database. The TM database returns an acknowledgment 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 *Application Development 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 "Understanding the Two-Phase Commit Process" on page 478 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 phase, the client sends a rollback message to all participating databases that successfully prepared to commit in the first phase. Each database then writes an "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* (refer to "Configuring DB2" in *Administration Guide, Performance*) 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 it 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 query the status of the transaction manager database for the status of this transaction 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.

If the transaction was coordinated by a Transaction Processing Monitor (XA-compliant transaction manager), then the database will always depend on the Transaction Processing Monitor to initiate the resynchronization.

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, refer to the *Command Reference* and the *Administrative 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 WITH PROMPTING 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 WITH PROMPTING 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 the LIST INDOUBT TRANSACTIONS WITH PROMPTING 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 the LIST INDOUBT TRANSACTIONS WITH PROMPTING 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 the LIST INDOUBT TRANSACTIONS WITH PROMPTING 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 the LIST INDOUBT TRANSACTIONS WITH PROMPTING 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, see "Other Configuration Considerations in Any Environment" on page 474.

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 Transactions on the Host

If your application has accessed a host or AS/400 database server within a transaction, there are some differences in how indoubt transactions are recovered.

To access host or AS/400 database servers, DB2 Connect is used. The recovery steps differ if DB2 Connect has the DB2 Syncpoint Manager configured.

Recovery when DB2 Connect Has the DB2 Syncpoint Manager Configured

The recovery of indoubt transactions at host or AS/400 servers is normally performed automatically by the Transaction Manager (TM) and the DB2 Syncpoint Manager (SPM). An indoubt transaction at a host or AS/400 server does not hold any resources at the local DB2 location, but does hold resources at the host or AS/400 server as long as the transaction is indoubt at that location. If the administrator of the host or AS/400 server determines that a heuristic decision must be made, then the administrator may contact the local DB2 database administrator (for example via telephone) to determine whether to commit or roll back the transaction at the host or AS/400 server. If this occurs, the LIST DRDA INDOUBT TRANSACTIONS command can be used to determine the state of the transaction at the DB2 Connect instance. The following steps can be used as a guideline for most situations involving an 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 host or AS/400 server. The partner_lu is the fully qualified luname of the host or AS/400 server. This provides the best identification of the host or AS/400 server, and should be provided by the caller from the host or AS/400 server. The luwid provides a unique identifier for a transaction and is available at all hosts and AS/400 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). If you issue the LIST DRDA INDOUBT TRANSACTIONS command with the WITH PROMPTING parameter, you can commit, roll back, or forget the transaction interactively. For more information, refer to the *Command Reference*.

```
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″, the SPM doesn't know whether the transaction is to be committed or rolled back. In this situation, you should use the WITH PROMPTING parameter to either commit or roll back the transaction on the DB2 Connect workstation. Then allow DB2 Connect to resynchronize with the host or AS/400 server based on the heuristic decision.

Recovery when DB2 Connect Does Not Use the DB2 Syncpoint Manager

Use the information in this section when TCP/IP connectivity is used to update DB2 for OS/390 in a multisite update from either DB2 Connect Personal Edition or DB2 Connect Enterprise Edition, and the DB2 Syncpoint Manager is not used. The recovery of indoubt transactions in this situation differs from that for indoubt transactions involving the DB2 Syncpoint Manager. When an indoubt transaction occurs in this environment, an alert entry is generated at the client, at the database server, and (or) at 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, refer to 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. You should allow automatic resynchronization to occur rather than heuristically force a decision at the database server. If, however, you must do this then use the following steps as a guideline.

Note: Because the DB2 Syncpoint Manager is not involved, you cannot use the LIST DRDA INDOUBT TRANSACTIONS command.

1. On the OS/390 host, issue the command DISPLAY THREAD TYPE(INDOUBT).

   From this list identify the transaction that you want to heuristically complete. Refer to the *DB2 for OS/390 Command Reference* for details of the DISPLAY command. The LUWID displayed can be matched to the same luwid at the Transaction Manager Database.

2. Issue the RECOVER THREAD(<LUWID>) ACTION(ABORT|COMMIT) command, depending on what you want to do.

   Refer to the *DB2 for OS/390 Command Reference* for details of the RECOVER command.

Chapter 11. 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 469.

The following topics are provided to assist you in using the database manager with an XA-compliant transaction manager such as IBM TXSeries CICS, IBM TXSeries Encina, BEA Tuxedo, or Microsoft Transaction Server:

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<,userid<,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, depending on the authentication method, are used to provide authentication information to the database.

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 10. Distributed Databases" on page 465. 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. You must have, at a minimum, the DB2 Client Application Enabler installed on the same machine as the XA Transaction Manager. Database servers that are accessed by applications controlled by the XA Transaction Manager can be either local or remote.

Updating Host or AS/400 Database Servers

Host and AS/400 database servers may be updatable depending upon the architecture of the XA Transaction Manager. If the work and the commit sequence occur within the same DB2 context (typically the same operating system thread), and the work is committed before starting a new transaction, then host and AS/400 database servers can participate in the transaction. Refer to the *Application Development Guide* for information about the SQL statements that are allowed in this environment.

If you will be updating host or AS/400 database servers, you will require DB2 Connect with the DB2 Syncpoint Manager configured. Refer to the *DB2 Connect Enterprise Edition for OS/2 and Windows NT Quick Beginnings* manual for instructions.

Database Connection Considerations

The sections that follow describe the database connection considerations:

- "RELEASE Statement"
- "Transactions Accessing Partitioned Databases (DB2 UDB EEE)"

RELEASE Statement

If 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.

Transactions Accessing Partitioned Databases (DB2 UDB EEE)

In a partitioned database environment, user data may be partitioned across database partitions. An application accessing the database connects and sends requests to one of the database partitions (the coordinator node). Different applications can connect to different database partitions, and the same application can choose different database partitions for different connections.

For a given transaction executing against a database in a partitioned environment, all access must be through the *same* database partition. That is, the same database partition must be used from the start of the transaction until (and including) the time that the transaction is committed.

Any transaction executing against the partitioned database must be committed before disconnecting.

Making a Heuristic Decision

An XA-compliant transaction manager (Transaction Processing Monitor) 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 478 . The primary difference between the two environments is that the TP Monitor provides the function of logging and controlling the transaction, instead of the DB2 transaction manager and the transaction manager database.

Errors similar to those discussed for the DB2 transaction manager (see "Recovering from Problems During Two-Phase Commit" on page 481) 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, refer to the *Command Reference* and the *Administrative 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 727, 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] 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.

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.

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:

- *tp_mon_name*

  The *tp_mon_name* database configuration parameter identifies the name of the transaction processor (TP) monitor product being used. For example, ″CICS″ or ″ENCINA″.

- *tpname*

  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. Refer to the *Quick Beginnings* manuals for more information.

- *tm_database*

  Because DB2 does *not* coordinate transactions in the XA environment, this parameter is not used for XA-coordinated transactions.

- *maxappls*

  The *maxappls* database configuration parameter specifies the maximum number of active applications allowed.

  The value of this parameter must be equal to or greater than the sum of the connected applications plus the number of these same applications which may be concurrently in the process of completing a two-phase commit or rollback. This sum should then have added to it the anticipated number of indoubt transactions which might exist at any one time. See "Recovering from Problems During Two-Phase Commit" on page 481 for more information on indoubt transactions.

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.

- *autorestart*

  This database configuration parameter specifies whether the RESTART DATABASE routine will be invoked automatically 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 Function Supported

DB2 Universal Database supports 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. DB2 UDB implements dynamic registration which is more advanced and efficient. Refer to "Resource Managers (RM)" on page 730 for more details about these two methods.

- Association Migration

  DB2 Universal Database does not support transaction migration between threads of control.

XA Switch Usage and Location

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 DB2 UDB uses dynamic registration and the TM should not use association migration. Also implicitly states that asynchronous operation is not supported.

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.

Using the DB2 Universal Database XA Switch

The XA architecture requires that a Resource Manager (RM) provide a *switch* that gives the XA Transaction Manager (TM) access to the resource manager's *xa_* routines. An RM's switch uses a structure called *xa_switch_t*. The switch contains the RM's name, non-null pointers to the RM's xa entry points, a flag, and a version number.

See the following sections for information on how to use the switch on different platforms:

- "UNIX Platforms" on page 498
- "OS/2 Platform" on page 498
- "Windows NT Platform" on page 498.

For a C sample program, see “Example C Code” on page 499.

UNIX Platforms: DB2 UDB's switch can be obtained in any of the following 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 a use of db2xa_switch.
- By calling db2xacic

  DB2 UDB 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( )
  ```

With either method, you must link your application with libdb2.

OS/2 Platform: DB2 UDB's switch can be obtained in any of the following 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 a use of db2xa_switch.
- Calling db2xacic

  DB2 UDB 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( )
  ```

With either method, you must link your application with db2api.lib.

Windows NT Platform: The interface to the db2xa_switch data structure is different for DB2 UDB for Windows NT because of 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 three 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 a 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 UDB 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( )
```

With any of these methods, ensure that you link with db2api.lib.

Example C Code: The following code illustrates the different ways the db2xa_switch can be accessed via a C program: on any UDB platform. Be sure to link with the appropriate library previously specified.

```
#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 Transaction Manager Known to DB2 Universal Database

DB2 must 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 and Windows NT, DB2 UDB must explicitly load the dynamic link library (DLL) that exports both these entry points at runtime. To accomplish this, the DLL name and path are retrieved from the *tp_mon_name* database manager configuration parameter.

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, refer to 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 51. Error Log for XA Open Error

Configuring XA Transaction Managers to Use DB2 UDB

The sections that follow describe how to configure specific products to use DB2 as a resource manager. You can use any of the following:

- "Configuring IBM TXSeries CICS" on page 501
- "Configuring IBM TXSeries Encina" on page 501
- "Configuring BEA Tuxedo" on page 504
- "Configuring Microsoft Transaction Server" on page 505.

Configuring IBM TXSeries CICS

For information about how to configure IBM TXSeries CICS to use DB2 as a resource manager, refer to your *IBM TXSeries CICS Administration Guide*. TXSeries documentation can be viewed online at starting at http://www.transarc.com/dfs/public/www/htdocs/.hosts/external/Library/index.html

Host and AS/400 database servers can participate in CICS-coordinated transactions.

Configuring IBM TXSeries Encina

The following are the various API and configuration parameters required for the integration of Encina Monitor and DB2 Universal Database servers or DB2 for MVS, DB2 for OS/390, DB2 for AS/400, or DB2 for VSE&VM when accessed via DB2 Connect. TXSeries documentation can be viewed online starting at http://www.transarc.com/dfs/public/www/htdocs/.hosts/external/Library/index.html

Host and AS/400 database servers can participate in Encina-coordinated transactions.

Configuring DB2

To configure DB2:

1. Each database name must be defined in the DB2 database directory. If the database is a remote database, then a Node Directory entry must also be defined. You can perform the configuration using the GUI Client Configuration Assistant (CCA), or the DB2 Command Line Processor (CLP). For example:

   ```
   DB2 CATALOG DATABASE inventdb AS inventdb AT NODE host1 AUTH SERVER

   DB2 CATALOG TCPIP NODE host1 REMOTE hostname1 SERVER svcname1
   ```

2. The DB2 client can optimize its internal processing for Encina if it knows that it is dealing with Encina. You can specify this by setting the *tp_mon_name* database manager configuration parameter to ENCINA. The default is for no special optimization. If *tp_mon_name* is set, then the application must ensure the thread that performs the unit of work also immediately commits the work after ending it. No other unit of work may be started. If this is *not* your environment, then ensure that the value for *tp_mon_name* value is NONE (or via the CLP, the value is set to NULL). The *tp_mon_name* can be updated by invoking the CCA or by the CLP:
   - On AIX use: `UPDATE DATABASE MANAGER CONFIGURATION USING TP_MON_NAME ENCINA`

- On Windows NT use: `UPDATE DATABASE MANAGER CONFIGURATION USING TP_MON_NAME libEncServer:E`

  In Intel environments, this parameter contains the path and name of the DLL in an external transaction manager product containing the functions ax_reg and ax_unreg, and also informs DB2 which TP Monitor is being used.

Configuring Encina for Each Resource Manager

To configure Encina for each resource manager, an administrator must define the Open String, Close String, and Thread of Control Agreement for each DB2 database as a resource manager before the resource manager can be registered for transactions in an application. The configuration can be performed using the Enconcole full screen interface, or the Encina command line interface. For example:

```
monadmin create rm inventdb  -open "inventdb,user1,password1"
```

There is one resource manager configuration for each DB2 database, and each resource manager (RM) configuration must have an rm name (″logical RM name″). To simplify the situation, you should make it identical to the database name.

The XA Open String contains information that is required to establish a connection to the database. The content of the string is RM specific. The XA Open String of DB2 UDB contains the alias name of the database to be opened, and optionally a userID and password to be associated with the connection. Note that the database name defined here must also be cataloged into the regular database directory required for all database access. The name can be up to 8 bytes long.

The XA Close String is not used by DB2.

The Thread of Control Agreement determines if an application agent thread can handle more than one transaction at a time. DB2 V5.0 and following supports the default of TMXA_SERIALIZE_ALL_OPERATIONS, where a thread can be reused only after a transaction has completed.

If you are accessing DB2 for OS/390, DB2 for MVS, DB2 for AS/400, or DB2 for VSE&VM, then you must use the DB2 Syncpoint Manager. Please refer to the *DB2 Connect Enterprise Edition for OS/2 and Windows NT Quick Beginnings* manual for configuration instructions.

Referencing a DB2 Database from an Encina Application

To reference a DB2 database from an Encina application:

1. Use the Encina Scheduling Policy API to specify how many application agents can be run from a single TP Monitor application process. For example:

```
rc = mon_SetSchedulingPolicy (MON_EXCLUSIVE)
```

   For DB2 (DB2 Universal Database, host, or AS/400 database servers), you should use the default setting of MON_EXCLUSIVE. This ensures that:

   - The application process is locked during the life time of the transaction.
   - The application acts single threaded.

   Note: If you are using the ODBC or DB2 Call Level Interface, you must disable the multithread support. You can do this by setting the CLI configuration parameter `DISABLEMULTITHREAD = 1` (disables multithreading). The default for DB2 Universal Database is `DISABLEMULTITHREAD = 0` (enables multithreading). Refer to the *CLI Guide and Reference* for more information.

2. Use the Encina RM Registration API to provide the XA switch and the logical RM name to be used by Encina when referencing the RM in an application process. For example:

```
rc = mon_RegisterRmi ( &db2xa_switch, /* xa switch */
                         "inventdb", /* logical RM name */
                       &rmiId  ); /* internal RM id */
```

   The XA Switch contains the addresses of the XA routines in the RM that the TM can call, and it also specifies the functionality that is provided by the RM. The XA Switch of DB2 Universal Database is `db2xa_switch`, and it resides in the DB2 client library (`db2app.dll` on INTEL platforms and `libdb2` on UNIX-based platforms).

   The logical RM name is the one used by Encina, and is not the actual database name that is used by the SQL application that runs under Encina. The actual database name is specified in the XA Open String in the Encina RM Registration. To simplify the situation, the logical RM name is set to be the same as the database name in this example.

   The third parameter returns an internal identifier or handle that is used by the TM to reference this connection.

Note: When using Encina for transaction processing with DB2 through the TM-XA interface, note that Encina nested transactions are not currently supported by the DB2 XA interface. If possible, avoid using these transactions. If you cannot, ensure that SQL work is done in only one member of the Encina transaction family.

Configuring BEA Tuxedo

Note: Applications that access host or AS/400 database servers in a Tuxedo environment are limited to read-only access to these servers.

To configure Tuxedo to use DB2 as a resource manager, perform the following steps:

1. Install Tuxedo as specified in the documentation for that product. Ensure that you perform all basic Tuxedo configuration, including the log files and environment variables.

   You also require a compiler and the DB2 Software Developer's Kit. Install these if necessary.
2. At the Tuxedo server ID, set the DB2INSTANCE environment variable to reference the instance that contains the databases that you want Tuxedo to use. Also set the PATH variable to include the DB2 program directories. Then confirm that the Tuxedo server ID can connect to the DB2 databases.
3. For Windows NT only. Update the *tp_mon_name* database manager configuration parameter with the name of the DLL that contains the `ax_reg` and `ax_unreg` routines. In Tuxedo, this DLL is called `libtux`.
4. Add a definition for DB2 to the Tuxedo resource manager definition file. In the examples that follow, `UDB_XA` is the locally defined Tuxedo resource manager name for DB2, and `db2xa_switch` is the DB2-defined name for a structure of type `xa_switch_t`:
   - For AIX. In the file ${TUXDIR}/udataobj/RM, add the definition:

     ```
     # DB2 UDB
     UDB_XA:db2xa_switch:-L${DB2DIR} /lib -ldb2
     ```

     Where {TUXDIR} is the directory where you installed Tuxedo, and {DB2DIR} is the DB2 instance directory.
   - For Windows NT. In the file %TUXDIR%\udataobj\rm, add the definition:

     ```
     # DB2 UDB
     UDB_XA;db2xa_switch;%DB2DIR%\lib\db2api.lib
     ```

     Where %TUXDIR% is the directory where you installed Tuxedo, and %DB2DIR% is the DB2 instance directory.
5. Build the Tuxedo transaction monitor server program for DB2:
   - For AIX:

     ```
     ${TUXDIR}/bin/buildtms -r UDB_XA -o ${TUXDIR}/bin/TMS_UDB
     ```

     Where {TUXDIR} is the directory where you installed Tuxedo.
   - For Windows NT:

     ```
     %TUXDIR%\bin\buildtms -r UDB_XA -o %TUXDIR%\bin\TMS_UDB
     ```

6. Build the application servers. In the examples that follow, the `-r` option specifies the resource manager name, the `-f` option (used one or more times) specifies the files that contain the application services, the `-s` option specifies the application service names for this server, and the `-o` option specifies the output server file name:
   - For AIX:

     ```
     ${TUXDIR}/bin/buildserver -r UDB_XA -f svcfile.o -s SVC1,SVC2 -o UDBserver
     ```

     Where {TUXDIR} is the directory where you installed Tuxedo.
   - For Windows NT:

     ```
     %TUXDIR%\bin\buildserver -r UDB_XA -f svcfile.o -s SVC1,SVC2 -o UDBserver
     ```

     Where %TUXDIR% is the directory where you installed Tuxedo.
7. Set up the Tuxedo configuration file to reference the DB2 server. In the *GROUPS section of the UDBCONFIG file, add an entry similar to:

   ```
   UDB_GRP   LMID=simp GRPNO=3
     TMSNAME=TMS_UDB TMSCOUNT=2
     OPENINFO="UDB_XA:SAMPLE,db2_user,,db2_user_pwd"
   ```

   Where the TMSNAME parameter specifies the transaction monitor server program that you built previously, and the OPENINFO parameter specifies the resource manager name. This is followed by the database name and the DB2 user and password, which are used for authentication.

   The application servers that you built previously are referenced in the *SERVERS section of the Tuxedo configuration file.
8. Start Tuxedo:

   ```
   tmboot -y
   ```

   After the command completes, Tuxedo messages should indicate that the servers are started. In addition, if you issue the DB2 command LIST APPLICATIONS ALL, you should see two connections (in this situation, specified by the TMSCOUNT parameter in the UDB group in the Tuxedo configuration file, UDBCONFIG.

Configuring Microsoft Transaction Server

DB2 UDB V5.2 and following can be fully integrated with Microsoft Transaction Server (MTS) Version 2.0. Applications running under MTS on Windows 32-bit operating systems can use MTS to coordinate two-phase commit with multiple DB2 UDB, host, and AS/400 database servers, as well as with other MTS-compliant resource managers.

Enabling MTS Support in DB2

Microsoft Transaction Server support is automatically enabled. While you can set the *tp_mon_name* database manager configuration parameter to "MTS", it is not necessary and will be ignored.

Note: Additional technical information may be provided on the IBM web site to assist you with installation and configuration of DB2 MTS support. Set your URL to ″http://www.software.ibm.com/data/db2/library″, and search for a DB2 Universal Database ″Technote″ with the keyword ″MTS″.

MTS Software Prerequisites

MTS support requires the DB2 Client Application Enabler (CAE) Version 5.2, or higher, and MTS must be at Version 2.0 with Hotfix 0772 or later releases.

The installation of the DB2 ODBC driver on Windows 32-bit operating systems will automatically add a new keyword into the registry:

```
HKEY_LOCAL_MACHINE\software\ODBC\odbcinit.ini\IBM DB2 ODBC Driver:
Keyword Value Name: CPTimeout
Data Type: REG_SZ
Value: 60
```

Installation and Configuration

Following is a summary of installation and configuration considerations for MTS. To use DB2's MTS support, the user must:

1. Install MTS and the DB2 client on the same machine where the MTS application runs.
2. If host or AS/400 database servers are to be involved in a multisite update:
   a. Install DB2 Connect Enterprise Edition either on your local machine or on a remote machine. DB2 Connect Enterprise Edition allows host or AS/400 database servers to participate in a multisite update transaction.
   b. Ensure your DB2 Connect Enterprise Edition Server is enabled for multisite update. For information on enabling DB2 Connect for multisite updates please see the DB2 Connect Enterprise Edition Quick Beginnings manual for your platform.

When running DB2 CLI/ODBC applications the following configuration keywords (as set in the `db2cli.ini` file) must not be changed from their default values:

- CONNECTYPE keyword (default 1)

- MULTICONNECT keyword (default 1)
- DISABLEMULTITHREAD keyword (default 1)
- CONNECTIONPOOLING keyword (default 0)
- KEEPCONNECTION keyword (default 0)

DB2 CLI applications written to make use of MTS support must not change the attribute values corresponding to the above keywords. In addition, the application must not change the default values of the following attributes:

- SQL_ATTR_CONNECT_TYPE attribute (default SQL_CONCURRENT_TRANS)
- SQL_ATTR_CONNECTON_POOLING attribute (default SQL_CP_OFF)

Note: Additional technical information may be provided on the IBM web site to assist you with installation and configuration of DB2 MTS support. Set your URL to ″http://www.software.ibm.com/data/db2/library″, and search for a DB2 Universal Database ″Technote″ with the keyword ″MTS″.

Verifying the Installation

1. Configure DB2 client and DB2 Connect EE to access your DB2 UDB, host, or AS/400 server.
2. Verify the connection from the DB2 CAE machine to the DB2 UDB database servers.
3. Verify the connection from the DB2 Connect machine to your host or AS/400 database server with DB2 CLP and issue a few queries.
4. Verify the connection from the DB2 CAE machine via the DB2 Connect gateway to your host or AS/400 database server and issue a few queries.

Supported DB2 Database Servers

The following servers are supported for multi-site update using MTS-coordinated transactions:

- DB2 Universal Database Enterprise Edition Version 5.2
- DB2 Enterprise - Extended Edition Version 5.2
- DB2 for OS/390
- DB2 for MVS
- DB2 for AS/400
- DB2 for VM&VSE
- DB2 Common Server for SCO, Version 2
- DB2 Universal Database for AIX with PTF U453782
- DB2 Universal Database for HP-UX with PTF U453784
- DB2 Universal Database Enterprise Edition for OS/2 with PTF WR09033

- DB2 Universal Database for SOLARIS with PTF U453783
- DB2 Universal Database Enterprise Edition for Windows NT with PTF WR09034
- DB2 Universal Database Extended Enterprise Edition for UNIX or Windows NT.

MTS Transaction Time-Out and DB2 Connection Behavior

You can set the transaction time-out value in the MTS Explorer tool. Please refer to the on-line *MTS Administrator Guide* for more details.

If a transaction takes longer than the transaction time-out value (default is 60 seconds), MTS will asynchronously issue an abort to all Resource Managers involved, and the whole transaction is aborted.

For the connection to a DB2 server, the abort is translated into a DB2 rollback request. Like any other database requests, the rollback request will be serialized on the connection to guarantee the integrity of the data on the database server.

As a result:

- If the connection is idle, the rollback is executed immediately.
- If a long running SQL statement is being executed, the rollback request will wait until the SQL statement finished before it is executed.

Connection Pooling

Connection pooling enables an application to use a connection from a pool of connections, so that the connection does not need to be reestablished for each use. Once a connection has been created and placed in a pool, an application can reuse that connection without performing a complete connection process. The connection is pooled when the application disconnects from the ODBC data source, and will be given to a new connection whose attributes are the same.

Connection pooling has been a feature of ODBC driver Manager 2.x. With the latest ODBC driver manager (version 3.5) that was shipped with MTS, connection pooling has some configuration changes and new behavior for ODBC connections of transactional MTS COM objects (see “Reusing ODBC Connections Between COM Objects Participating in the Same Transaction” on page 509).

ODBC driver Manager 3.5 requires that the ODBC driver register a new keyword in the registry before it allows connection pooling to be activated. The keyword is:

```
Key Name: SOFTWARE\ODBC\ODBCINST.INI\IBM DB2 ODBC DRIVER
Name: CPTimeout
Type: REG_SZ
Data: 60
```

The DB2 ODBC driver version 6 and later for 32-bit Windows operating system fully supports connection pooling and therefore this keyword is registered. Version 5.2 clients must install Fix Pack 3 (WR09024) or later.

The default value (60) means the connection will be pooled for 60 seconds before it actually is disconnected.

In a busy environment, it is better to increase the CPTimeout value to a large number (Microsoft sometimes suggests 10 minutes for certain environments) to prevent too many physical connects and disconnects, because these consume a large amount of system resources, including system memory and communications stack resource.

Reusing ODBC Connections Between COM Objects Participating in the Same Transaction

ODBC connections in MTS COM objects have connection pooling turned on automatically (whether or not the COM object is transactional) .

For multiple MTS COM objects participating in the same transaction, the connection can be reused between two or more COM objects in the following manner.

Suppose there are two COM objects, COM1 and COM2 that connect to the same ODBC datasource and participate in the same transaction.

After COM1 connects and does its work, it disconnects and the connection is pooled. However, this connection will be reserved for the use of other COM objects of the same transaction. It will be available to other transactions only after the current transaction ends.

When COM2 is invoked in the same transaction, it is given the pooled connection. MTS will ensure that the connection can only be given to the COM objects that are participating in the same transaction.

On the other hand, if COM1 does not explicitly disconnect, then it will tie up the connection until the transaction ends. When COM2 is invoked in the same transaction, a separate connection will be acquired. Subsequently, this transaction ties up two connections instead of one.

This reuse of connection feature for COM objects participating in the same transaction is preferable for the following reasons:

- It uses fewer resources in both the client and the server. Only one connection is needed.
- It eliminates the possibility that two connections participating in the same transaction (accessing the same database server and accessing the same data) can lock one another, because DB2 servers treat different connections from MTS COM objects as separate transactions.

Tuning TCP/IP Communications

If a small CPTimeout value is used in a high-workload environment where too many physical connects and disconnects occur at the same time, the TCP/IP stack may encounter resource problems.

To alleviate this problem, you should use the TCP/IP Registry Entries. These are described in the *Windows NT Resource Guide*, Volume 1. The registry key values are located in HKEY_LOCAL_MACHINE-> SYSTEM-> CurrentControlSet-> Services-> TCPIP-> Parameters.

The default values and suggested settings are as follows:

Name	Default Value	Suggested Value
KeepAlive time	7200000 (2 hours)	Same
KeepAlive interval	1000 (1 second)	10000 (10 seconds)
TcpKeepCnt	120 (2 minutes)	240 (4 minutes)
TcpKeepTries	20 (20 re-tries)	Same
TcpMaxConnectAttempts	3	6
TcpMaxConnectRetransmission	3	6
TcpMaxDataRetransmission	5	8
TcpMaxRetransmissionAttempts	7	10
If the registry value is not defined, then create it.		

Testing DB2 With The MTS ″BANK″ Sample Application

You can use the ″BANK″ sample program that is shipped with MTS to test the setup of the client products and MTS.

Follow these steps:

1. Change the file `\Program Files\Common Files\ODBC\Data Sources\MTSSamples.dsn` so that it looks like this:

```
[ODBC]
DRIVER=IBM DB2 ODBC DRIVER
UID=your_user_id
PWD=your_password
DSN=your_database_alias
Description=MTS Samples
```

where:

- your_user_idand your_password are the user-ID and password used to connect to the host.
- your_database_alias is the database alias used to connect to the database server.

2. Go to ODBC Administration in the Control Panel, click on System DSN tab and add the data source:
   a. Choose IBM ODBC Driver and click on Finish.
   b. When presented with the list of database aliases, choose the one that was specified previously.
   c. Click on OK
3. Use DB2 CLP to connect to a DB2 database under the ID your_user_id, as above.
   a. Bind the db2cli.lst:

      ```
      db2 bind @C:\sqllib\bnd\db2cli.lst blocking all grant public
      ```

   b. Bind the utilities.

      If the server is a DRDA host server, bind ddcsmvs.lst, ddcs400.lst, or ddcsvm.lst, depending on the host that you are connecting to (OS/390, AS/400, or VSE or VM). For example:

      ```
      db2 bind @C:\sqllib\bnd\@ddcsmvs.lst blocking all grant public
      ```

      Otherwise, bind the db2ubind.lst:

      ```
      db2 bind @C:\sqllib\bnd\@db2ubind.lst blocking all grant public
      ```

   c. Then create the sample table and data for the MTS sample application as follows:

      ```
      DB2 CREATE TABLE ACCOUNT (ACCOUNTNO INT, BALANCE INT)
      DB2 INSERT INTO ACCOUNT VALUES(1, 1)
      ```

4. On the DB2 client, ensure that the database manager configuration parameter *tp_mon_name* is set to ″MTS″.
5. Run the ″BANK″ application. Select the **Account** button and the **Visual C++** option, then submit the request. Other options may use SQL that is specific to SQL Server, and may not work.

Part 4. Ensuring the High Availability of Your System

Chapter 12. High Availability Cluster Multi-Processing (HACMP) on AIX

DB2 UDB provides high availability failover support on many platforms. On AIX, DB2 UDB supports failover through the capabilities of IBM High Availability Cluster Multi-Processing (HACMP). Failover capability allows for the automatic transfer of workload from one processor to another should there be a hardware failure.

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.

Note: Do not use a “kill -9” against the db2start process in a high availability environment. This action is not recommended in any environment, but in particular such an action may invalidate failover recovery in your high availability environment.

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/samples/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 52 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 522.

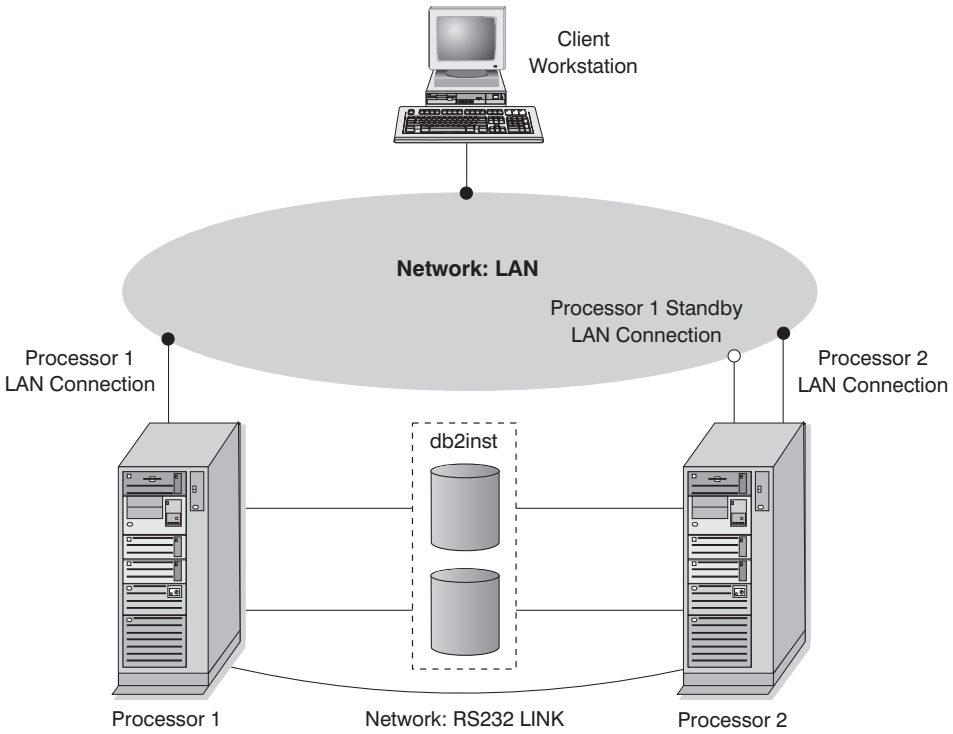


Figure 52. Instance Failover Example

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 366 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 hostname 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 hostname node202 port 0 netname node202
        db2start nodenum 3 restart hostname node202 port 1 netname node202
        db2start nodenum 4 restart hostname 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/samples/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”.

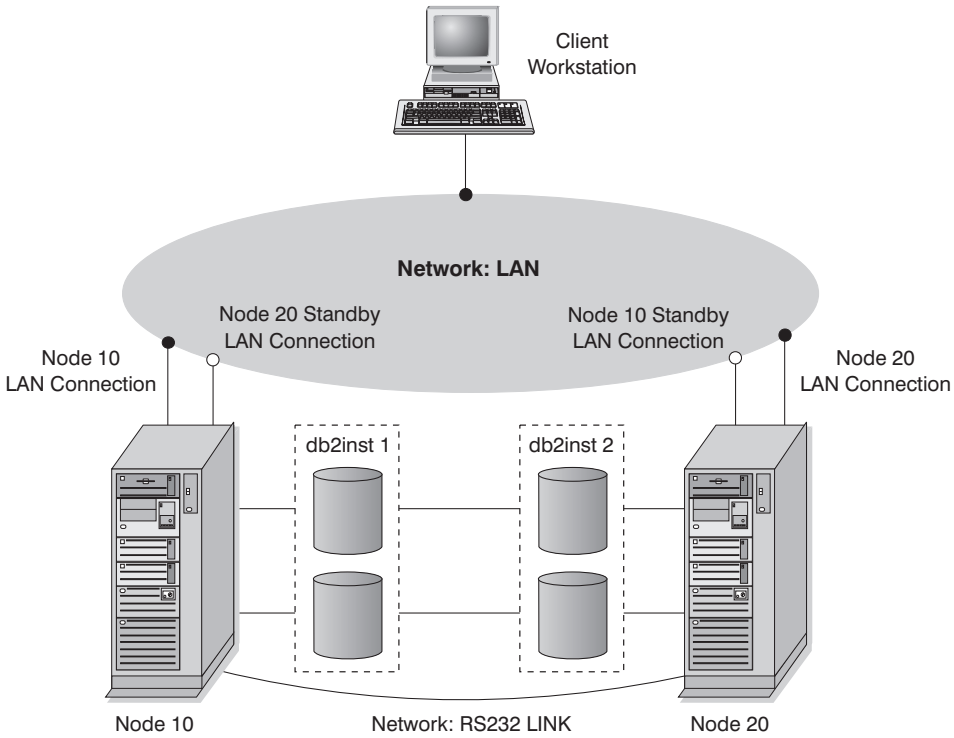


Figure 53. Instance Failover Example

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 hostname 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.

Chapter 13. High Availability Cluster Multi-Processing, Enhanced Scalability (HACMP ES) for AIX

Enhanced Scalability is a feature of HACMP for AIX Version 4.2.2 which currently only runs on RS/6000 SP nodes.

This feature provides the same failover recovery as HACMP and has identical event structure to previous HACMP versions. There are several documented differences to this event structure documented in the *HACMP for AIX, V4.2.2, Enhanced Scalability Installation and Administration Guide*. Beyond these standard items, the Enhanced Scalability feature provides:

- Larger HACMP clusters with scalability up to 16 nodes per cluster.
- Additional error coverage through "User-Defined Events". Monitored areas can trigger user-defined events which can be as diverse as the death of a process or the fact that paging space is nearing capacity. Once detected, events are triggered.

  Such events include pre- and post-events that can be added to the failover recovery process, if needed. Extra functions that are specific to the different implementations can be placed within the HACMP pre- and post-event streams.

  A rules file (`/usr/sbin/cluster/events/rules.hacmprd`) exists and contains the HACMP events. User-defined events are added to this file and the script files to be run when events occur are part of this definition. The rules file is described in more detail later.
- HACMP client utilities for monitoring and detecting status changes in one or more clusters from AIX physical nodes outside the HACMP cluster.
- Although not an enhancement, the discussion of HACMP ES concludes with an overview of the installation and migration planning required for this feature.

Note: Do not use a "kill -9" against the db2start process in a high availability environment. This action is not recommended in any environment, but in particular such an action may invalidate failover recovery in your high availability environment.

The nodes in HACMP ES clusters exchange messages called "heartbeats" or "keepalive" packets which inform the other nodes regarding the availability of each node in the cluster. A node that has stopped responding causes the remaining nodes in the cluster to invoke recovery. The recovery process is called a "node_down event" and may also be referred to as "failover". The

completion of the recovery process is followed by work done on the node that is down with the goal being the re-integration of the node into the cluster. This is called a “node_up event”.

There are two types of events: standard events that are anticipated within the operations of HACMP ES; and, user-defined events which are associated with the monitoring of parameters in hardware and software components.

One of the standard events is the node_down event. When planning what should be done as part of the recovery process, HACMP allows two failover options: Hot (or idle) Standby; and, Mutual Takeover.

Cluster Configuration

In a hot standby configuration, the AIX processor node that is the takeover node is not running any other workload. In a mutual takeover configuration, the AIX processor node that is the takeover node is running other workload.

Generally, DB2 UDB EEE runs in mutual takeover mode with partitions on each node. One exception is a scenario where the catalog node is part of a hot standby configuration.

When planning a large DB2 installation on a RS/6000 SP using HACMP ES, you need to consider how to divide the nodes of the cluster within or between the RS/6000 SP frames. Having a node and its backup in different SP frames can allow takeover in the event one frame goes down (that is, the frame power/switch board fails). However, such failures are expected to be exceedingly rare because there are N+1 power supplies in each SP frame and each SP switch has redundant paths along with N+1 fans and power. In the case of a frame failure, manual intervention may be required to recover the remaining frames. This recovery procedure is documented in the SP Administration Guide. HACMP ES provides for recovery of SP node failures; recovery of frame failures is dependent on proper layout of clusters within the SP frame(s).

Another planning consideration involves how to manage big clusters: It is easier to manage a small cluster than a big one; however, it is also easier to manage one big cluster than many smaller ones. When planning, consider how your applications will be used in your cluster environment. If there is a single, large, homogeneous application running on, for example, 16 nodes then it is probably easier to manage as a single cluster rather than as eight (8) two-node clusters. If the same 16 nodes contain many different applications with different networks, disks, and node relationships then it is probably better to group the nodes into smaller clusters. Keep in mind that nodes integrate into an HACMP cluster one at a time; it will be faster to start a

configuration of multiple clusters rather than one large cluster. HACMP ES supports both single and multiple clusters as long as a node and its backup are in the same cluster.

HACMP ES failover recovery allows pre-defined (also known as “cascading”) assignment of a resource group to a physical node. The failover recovery procedure also allows floating (also known as “rotating”) assignment of a resource group to a physical node. IP addresses; external disk volume groups, filesystems, NFS filesystems; and, application servers within each resource group specify either an application or application component which can be manipulated by HACMP ES between physical nodes by failover and reintegration. Failover and reintegration behavior is specified by the type of resource group created, and by the number of nodes placed in the resource group.

As an example, consider a DB2 database partition (logical node): If its log and table space containers were placed on external disks, and other nodes were linked to that disk, it would be possible for those other nodes to access these disks and restart the database partition (on a takeover node). It is this type of operation that is automated by HACMP. HACMP ES can also be used to recover NFS file systems used by DB2 instance main user directories.

Read the HACMP ES documentation thoroughly as part of your planning for recovery with DB2 UDB EEE. You should read the Concepts, Planning, Installation, and Administration guides. Then you can layout the recovery architecture for your environment. For the subsystems you have identified for recovery based on the identified points of failure, identify the HACMP clusters you need and the recovery nodes for each (either hot standby or mutual takeover). This architecture and planning is a starting point for completing the HACMP worksheets found in the documentation (mentioned above).

It is strongly recommended that both disks and adapters are mirrored in your external disk configuration. For DB2 physical nodes that are configured for HACMP, care is required to ensure that nodes can vary on the volume group from the shared external disks. In a mutual takeover configuration, this arrangement requires some additional planning so that the paired nodes can access each other’s volume groups without conflicts. Within DB2 UDB EEE this means that all container names must be unique across all databases.

One way to achieve uniqueness in the names is to include the partition number as part of the name. You can specify a node expression for container string syntax when creating either SMS or DMS containers. When you specify the expression, either the node number is part of the container name, or, if you specify additional arguments, the result of the argument is part of the container name. You use the argument “ $N” ([blank]$N) to indicate the node

expression. The argument must occur at the end of the container string and can only be used in one of the following forms. In the table below, the node number is assumed to be five (5):

Table 35. Arguments for Creating Containers

Syntax	Example	Value
[blank]$N	" $N"	5
[blank]$N+[number]	" $N+1011"	1016
[blank]$N%[number]	" $N%3"	2
[blank]$N+[number]%[number]	" $N+12%13"	4
[blank]$N%[number]+[number]	" $N%3+20"	22

Notes:

1. % is modulus.
2. In all cases, the operators are evaluated from left to right.

Following are some examples of creating containers using this special argument:

- Creating containers for use on a two-node system.

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
   (device '/dev/rcont $N' 20000)
```

  The following containers would be used:

```
/dev/rcont0   - on Node 0
/dev/rcont1   - on Node 1
```

- Creating containers for use on a four-node system.

```
CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
   (file '/DB2/containers/TS2/container $N+100' 10000)
```

  The following containers would be used:

```
/DB2/containers/TS2/container100   - on Node 0
/DB2/containers/TS2/container101   - on Node 1
/DB2/containers/TS2/container102   - on Node 2
/DB2/containers/TS2/container103   - on Node 3
```

- Creating containers for use on a two-node system.

```
CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
   ('/TS3/cont $N%2, '/TS3/cont $N%2+2')
```

  The following containers would be used:

```
/TS3/cont0   - on Node 0
/TS3/cont2   - on Node 0
/TS3/cont1   - on Node 1
/TS3/cont3   - on Node 1
```

The following pictures show some of the planning involved to ensure a highly available external disk configuration and the ability to access all volume groups without conflict.

DB2 SSA I/O Subsystem Configuration - No single point of failure

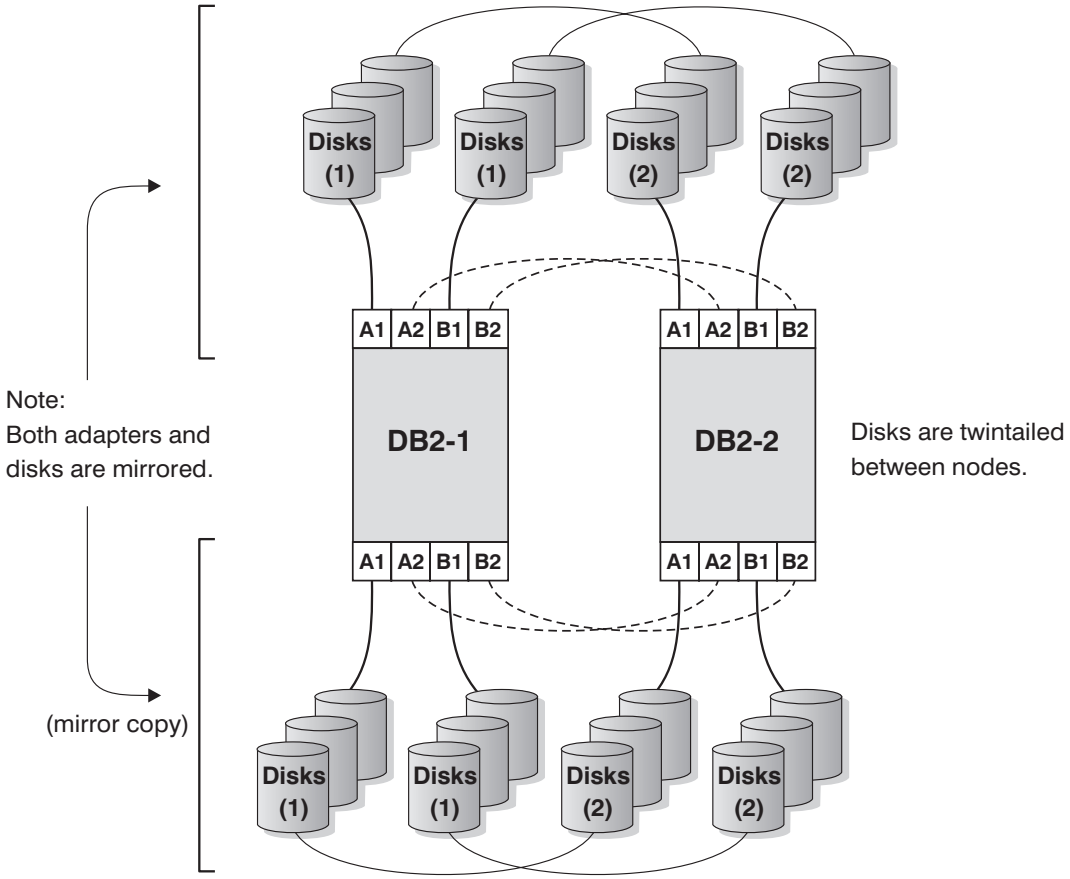


Figure 54. No Single Point of Failure

DB2 SSA I/O Subsystem Configuration - Volume group and logical volume setup

db2 database testdata on filesystem /database instance name powertp

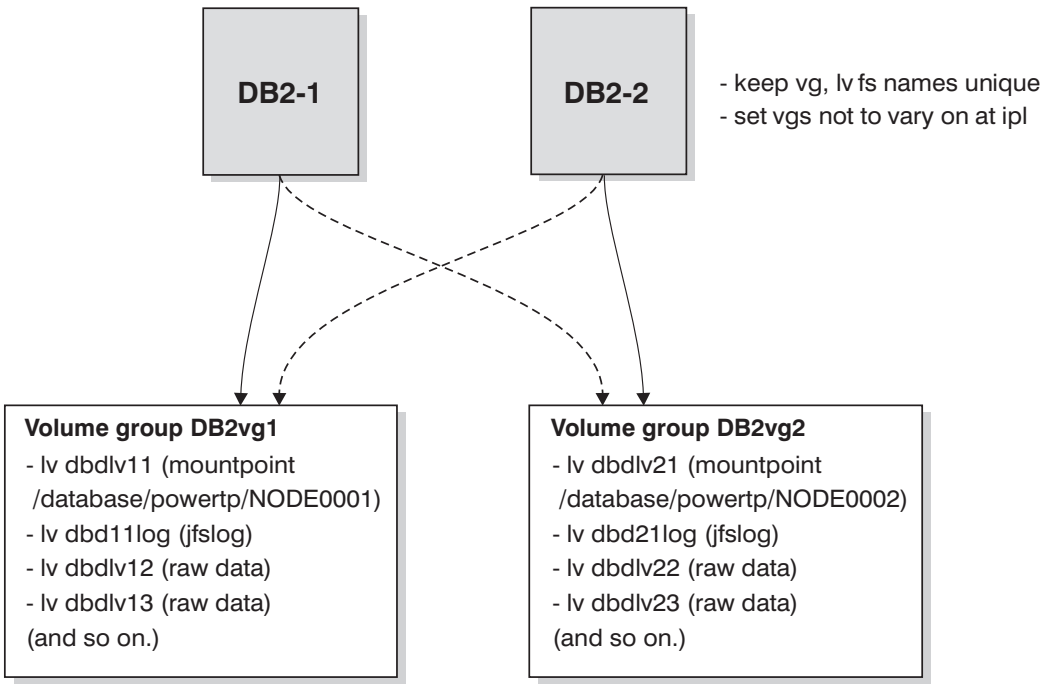


Figure 55. Volume Group and Logical Volume Setup

Once configured, each database partition in an instance is started by HACMP ES one physical node at a time. Using multiple clusters is recommended for starting parallel DB2 configurations that are larger than four (4) nodes.

Note: Each HACMP node in a cluster is started one at a time. For a 64-node parallel DB2 configuration, it is faster to start 32, two-node HACMP clusters in parallel rather than four (4), sixteen-node clusters.

A script file, `rc.db2pe`, is packaged with DB2 UDB EEE to assist in configuring for HACMP ES failover or recovery in either “hot standby” or “mutual takeover” nodes. In addition, DB2 buffer pool sizes can be customized during failover in mutual takeover configurations from within `rc.db2pe`. (Buffer pool size modification is needed to ensure proper performance when two database

partitions run on one physical node. See the next section for additional information.) The script file, rc.db2pe, is installed on each node in /usr/bin.

Configuration of a DB2 Database Partition

When you create an application server in a HACMP configuration of a DB2 database partition, specify rc.db2pe as a start and stop script in the following way:

```
/usr/bin/rc.db2pe <instance> <dpn> <secondary dpn> start <use switch>
/usr/bin/rc.db2pe <instance> <dpn> <secondary dpn> stop <use switch>
```

where:

```
<instance>  is the instance name.
<dpn> is the database partition number.
<secondary dpn> is the 'companion' database partition number in
   'mutual takeover' configurations only; in 'hot standby' configurations
   it is the same as <dpn>.
<use switch> is usually blank; when blank, by default this indicates that
   the SP Switch network is used for hostname field in the db2nodes.cfg file
   (all traffic for DB2 is routed over the SP switch); if not blank, the name
   used is the hostname of the SP node to be used.
```

Note: The DB2 command LIST DATABASE DIRECTORY is used from within rc.db2pe to find all databases configured for this database partition. The rc.db2pe script file then looks for /usr/bin/reg.parms.DATABASE and /usr/bin/failover.parms.DATABASE files, where DATABASE is each of the databases configured for this database partition. In a "mutual takeover" configuration, it is recommended you create these parameter files (reg.parms.xxx and failover.parms.xxx). In the failover.parms.xxx file, the settings for BUFFPAGE, DBHEAP, and any others affecting buffer pools should be adjusted to account for the possibility of more than one buffer pool. Buffer pool size modification is needed to ensure proper performance when two or more database partitions run on one physical node. Sample files reg.parms.SAMPLE and failover.parms.SAMPLE are provided for your use.

One of the important parameters in this environment is START_STOP_TIME. This database manager configuration parameter has a default value of ten (10) minutes. However, rc.db2pe sets this parameter to two (2) minutes. You should modify this parameter within rc.db2pe so that it is set to ten (10) minutes or perhaps something slightly larger. The length of time in the context of a failed database partition is the time between the failure of the partition and the recovery of that partition. If there are frequent "COMMIT"s used in the applications running on a partition, then ten minutes following the failure on a database partition should be sufficient time to rollback uncommitted transactions and reach a point of consistency for the database on that partition. If your workload is heavy and/or you have many partitions, you may need to increase the parameter value until there is no longer an

additional problem beyond that of the original partition failure. (The additional problem would be the timeout message resulting from exceeding the START_STOP_TIME value while waiting for the rollback to complete at the failed database partition.)

Example of a Mutual Takeover Configuration

The assumption in this example is that the mutual takeover configuration will exist between physical nodes one and two with a DB2 instance name of "POWERTP". The database partitions are one and two, and the database name is "TESTDATA" on filesystem /database.

```
Resource group name: db2_dp_1
Node Relationship: cascading
Participating nodenames: node1_eth, node2_eth
Service_IP_label: nfs_switch_1     (<<< this is the switch alias address)
Filesystems: /database/powertp/NODE0001
Volume Groups: DB2vg1
Application Servers: db2_dp1_app
Application Server Start Script: /usr/bin/rc.db2pe powertp 1 2 start
Application Server Stop Script: /usr/bin/rc.db2pe powertp 1 2 stop

Resource group name: db2_pd_2
Node Relationship: cascading
Participating nodenames: node2_eth, node1_eth
Service_IP_label: nfs_switch_2     (<<< this is the switch alias address)
Filesystems: /database/powertp/NODE0002
Volume Groups: DB2vg2
Application Servers: db2_dp2_app
Application Server Start Script: /usr/bin/rc.db2pe powertp 2 1 start
Application Server Stop Script: /usr/bin/rc.db2pe powertp 2 1 stop
```

Example of a Hot Standby Takeover Configuration

The assumption in this example is that the hot standby takeover configuration will exist between physical nodes one and two with a DB2 instance name of "POWERTP". The database partition is one, and the database name is "TESTDATA" on filesystem /database.

```
Resource group name: db2_dp_1
Node Relationship: cascading
Participating nodenames: node1_eth, node2_eth
Service_IP_label: nfs_switch_1     (<<< this is the switch alias address)
Filesystems: /database/powertp/NODE0001
Volume Groups: DB2vg1
Application Servers: db2_dp1_app
Application Server Start Script: /usr/bin/rc.db2pe powertp 1 1 start
Application Server Stop Script: /usr/bin/rc.db2pe powertp 1 1 stop
```

Note: In both examples, the resource groups contain a Service IP switch alias address. This switch alias address is used for:

1. NFS access to a file server for the DB2 instance owner filesystems.

2. Other client access that needs to be maintained in the case of a failover, ADSM connection, or other similar operations.

If your implementation does not require these aliases, they can be removed. If removed, be sure to set the MOUNT_NFS parameter to "NO" in rc.db2pe.

Configuration of a NFS Server Node

Just as with the configuration of a DB2 database partition presented above, the rc.db2pe script can be used to make available NFS-mounted directories of DB2 parallel instance user directories. This can be accomplished by setting the MOUNT_NFS parameter to "YES" in rc.db2pe and configuring the NFS failover server pair as follows:

- Configure the home directory and export it as "root" using /etc/exports and exportfs command to the IP address used on the nodes in the same subnet as the NFS Server's IP address. Include both the HACMP boot and service addresses. The NFS Server's IP address is the same address as the service address in HACMP that can be taken over by a backup. The home directory of the DB2 instance owner should be NFS-mounted directly, not automounted. (The use of the automounter is not supported by the scripts as a DB2 instance owner home directory.)
- Using SMIT or a bottom-line configuration, a separate /etc/filesystems entry should be created for this filesystem so that all nodes in the DB2 parallel grouping, including the file server, can mount using the NFS filesystem command.

  For example, an /nfshome JFS filesystem can be exported to all nodes as /dbhome. Each node creates a NFS filesystem /dbname which is nfs_server:/nfshome. Therefore, the home directory of the DB2 instance owner would be /dbhome/powertp when the instance name is "powertp".

  Ensure the NFS parameters for the mount in /etc/filesystems are "hard", "bg", "intr", and "rw".
- Ensure the DB2 instance owner definitions associated with the home directory /dbhome/powertp in /etc/passwd are the same on all nodes.

  The user definitions in an SP environment are typically created on the Control Workstation and "supper" or "pcp" is used to distribute /etc/passwd, /etc/security/passwd, /etc/security/user, and /etc/security/group to all nodes.
- Do NOT configure the "nfs_filesystems to export" in HACMP resource groups for the volume group and the filesystem that is exported. Instead, configure it normally to NFS. The scripts for the NFS server will control the exporting of the filesystems.
- Ensure the major number of the volume group where the filesystem resides is the same on both the primary node and the takeover node. This is accomplished by using importvg with the -V parameter.

- Verify that the MOUNT_NFS parameter is set to “YES” in rc.db2pe and that each node has the NFS filesystem to mount in /etc/filesystems. If this is not the case, then rc.db2pe will not be able to mount the filesystem and start DB2.
- If the DB2 instance owner was already created and you are copying the user’s directory structure to the filesystem you are creating, ensure you tar (-cvf) the directory. This ensures the preservation of the symbolic links.
- Do not forget to mirror both the adapters and the disks for the logical volumes and the filesystem logs of the filesystem you are creating.

Example of a NFS Server Takeover Configuration

The assumptions in this example are that there is an NFS server filesystem /nfshome in the volume group nfsvg over the IP address “nfs_server”. The DB2 instance name is “POWERTP” and the home directory is /dbhome/powertp.

```
Resource group name: nfs_server
Node Relationship: cascading
Participating nodenames: node1_eth, node2_eth
Service_IP_label: nfs_server     (<<< this is the switch alias address)
Filesystems: /nfshome
Volume Groups: nfsvg
Application Servers: nfs_server_app
Application Server Start Script: /usr/bin/rc.db2pe powertp NFS SERVER start
Application Server Stop Script: /usr/bin/rc.db2pe powertp NFS SERVER stop
```

Note: In this example:

- /etc/filesystems on all nodes would contain an entry for /dbhome as mounting nfs_server:/nfshome. nfs_server is a Service IP switch alias address.
- /etc/exports on the nfs_server node and the backup node would include the boot and service addresses and contain an entry for /nfsfs -root=nfs_switch_1, nfs_switch_2, ...

Considerations When Configuring the SP Switch

When implementing HACMP ES with the SP switch, consider the following:

- There are “base” and “alias” addresses on the SP switch. The base addresses are those defined in the SP System Data Repository (SDR), and are configured by rc.switch when the system is “booted”. The alias addresses are IP addresses configured, in addition to the base address, into the css0 interface through use of the ifconfig command with an alias attribute. For example:

  ```
  ifconfig css0 inet alias sw_alias_1 up
  ```

- When configuring the DB2 db2nodes.cfg file, SP switch “base” IP address names should be used for both “hostname” and “netname” fields. Switch IP

address aliases are ONLY used to maintain NFS connectivity. DB2 failover is achieved by restarting DB2 with the db2start restart command (which updates db2nodes.cfg).

- Do not confuse the switch addresses with the etc/hosts aliases. Both the SP switch addresses and the SP switch alias addresses are real in either etc/hosts or DNS. The switch alias addresses are not another name for the SP switch base address: Each has its own separate address.
- The SP switch base addresses are always present on a node when it is up. HACMP ES does not configure or move these addresses between nodes.
- If you intend to use SP switch alias addresses, configure these to HACMP as boot and service addresses for "heartbeating" and IP address takeover. If you do not intend to use SP switch alias addresses, configure the base SP switch address to HACMP as a service address for "heartbeating" **ONLY** (no IP address takeover). Do not, in any configuration, configure alias addresses **AND** the switch base address; this configuration is not supported by HACMP ES.
- Only the SP switch alias addresses are moved between nodes for an IP takeover configuration and not the SP switch base addresses.
- The need for SP switch aliases arises because there can only be one SP switch adapter per node. Using alias addresses allows a node to takeover another node's switch alias IP address without adding another switch adapter. This is useful in nodes that are "slot-constrained". For more information on handling recovery from SP switch adapter failures, see the network failure section under "HACMP ES Script Files" on page 547 later in this document.
- If you configure the SP switch for IP address takeover, you will need to create two (2) extra alias IP addresses per node: One as a boot address and one as a service address.
- Do not forget to use "HPS" in the HACMP ES network name definition for a SP switch base IP address or a SP switch alias IP address.
- rc.cluster in HACMP automatically ifconfigs-in the SP switch boot address when HACMP is started. No additional configuration is required other than the creating the IP address and name, and defining them to HACMP.
- The SP switch Eprimary node is moved between nodes by the SP Parallel System Support Program (PSSP), and not HACMP. If an Eprimary node goes offline, the PSSP automatically has a backup node assume responsibility as the Eprimary node. The switch network is unaffected by this change and remains up.
- The Eprimary node of the SP switch is the server that implements the Estart and Efence/Eunfence commands. The HACMP scripts attempt to Eunfence or to Estart a node when HACMP is started and make the switch available should it be defined as one of its networks. For this reason, ensure the

Eprimary node is available when you start HACMP. The HACMP code waits up to twelve (12) minutes for an Eprimary failover to complete before it exits with an error.

DB2 HACMP Configuration Examples

The following examples show different possible failover support configurations and what happens when failure occurs.

DB2 HACMP Mutual Takeover with NFS Failover - Normal

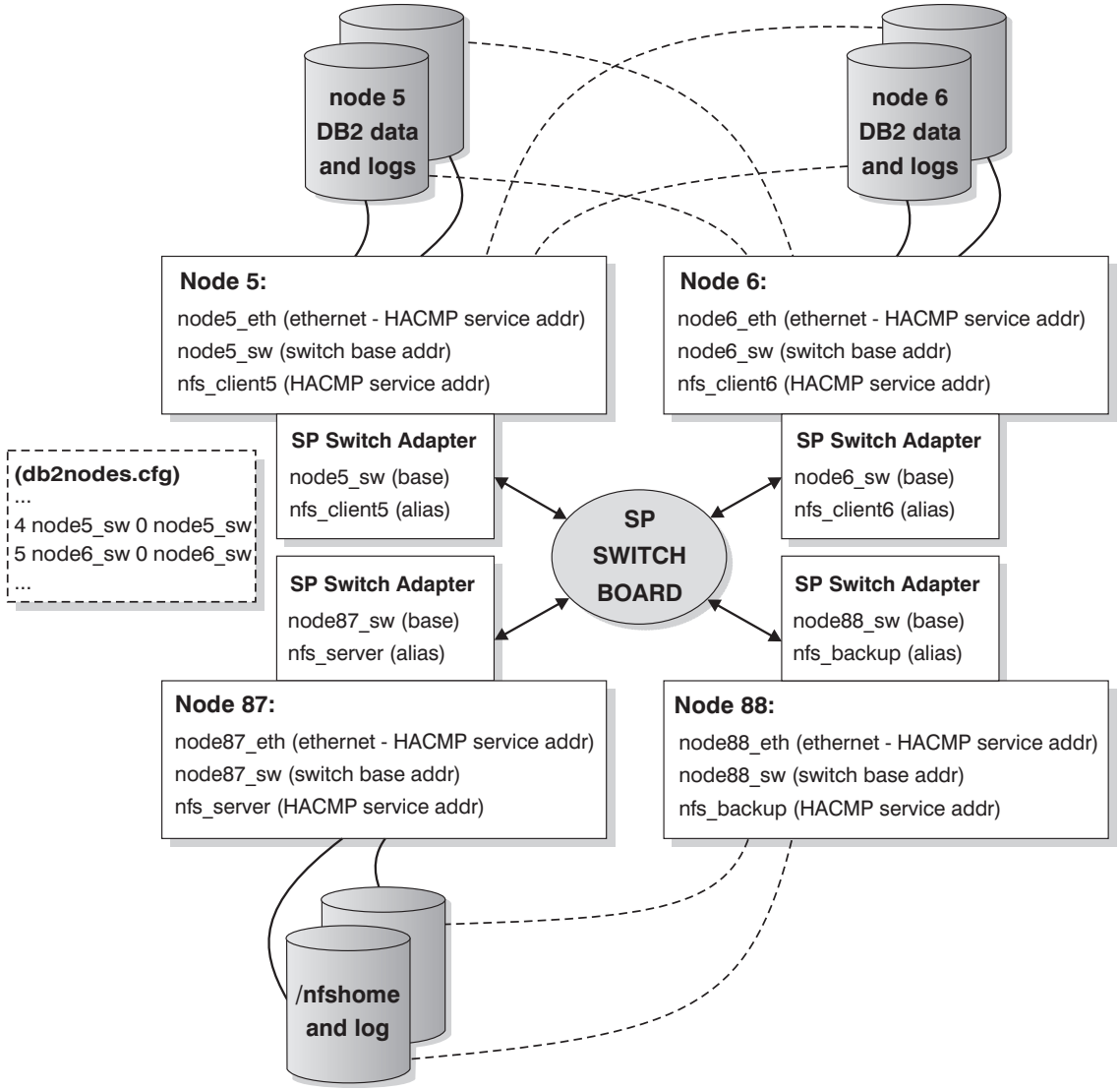


Figure 56. Mutual Takeover with NFS Failover - Normal

The previous figure and the next two figures each have the following notes associated with them:

1. HACMP adapters are defined for ethernet, and SP Switch alias boot and service aliases — base addresses are untouched. Remember to use a “HPS” string in the HACMP network name.

2. The NFS_server/nfshome is mounted as /dbhome on all nodes through switch aliases.
3. The db2nodes.cfg file contains SP Switch base addresses. The db2nodes.cfg file is changed by the DB2START RESTART command after a DB2 database partition (logical node) failover.
4. The Switch alias boot addresses are not shown.
5. Nodes can be in different SP frames.

DB2 HACMP Mutual Takeover with NFS Failover - NFS failover

- nfs_server SP Switch alias IP addr and nfs mounted /nfshome moved from node 87 to 88.
- SP switch arp code has functionality to update all switch arp caches with this move.

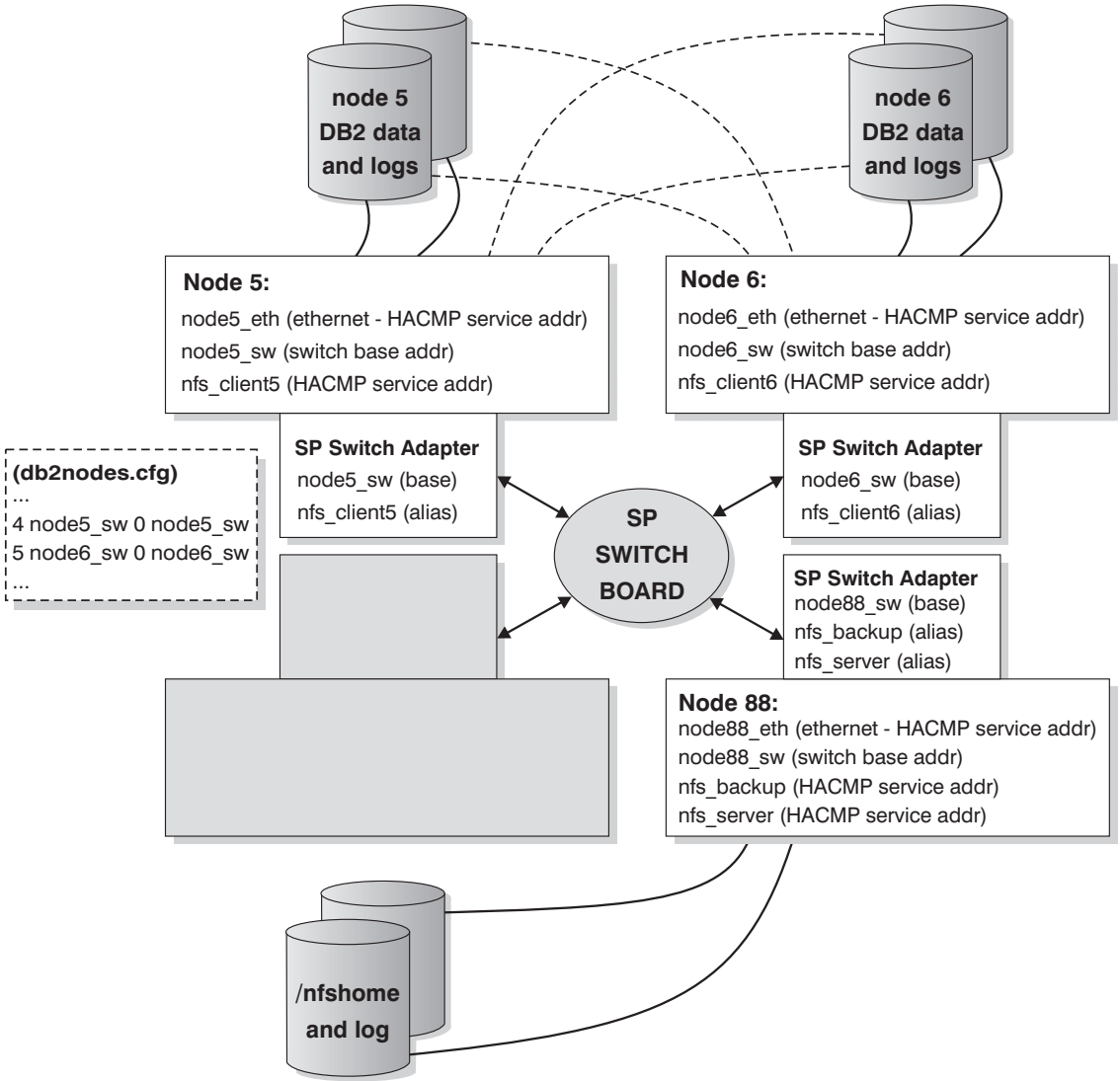


Figure 57. Mutual Takeover with NFS Failover - NFS Failover

DB2 HACMP Mutual Takeover with NFS Failover - DB2 failover

- switch IP address takeover allows other servers (like ADSM) to retain connectivity.
- Node 5 runs 2 logical nodes of DB2.

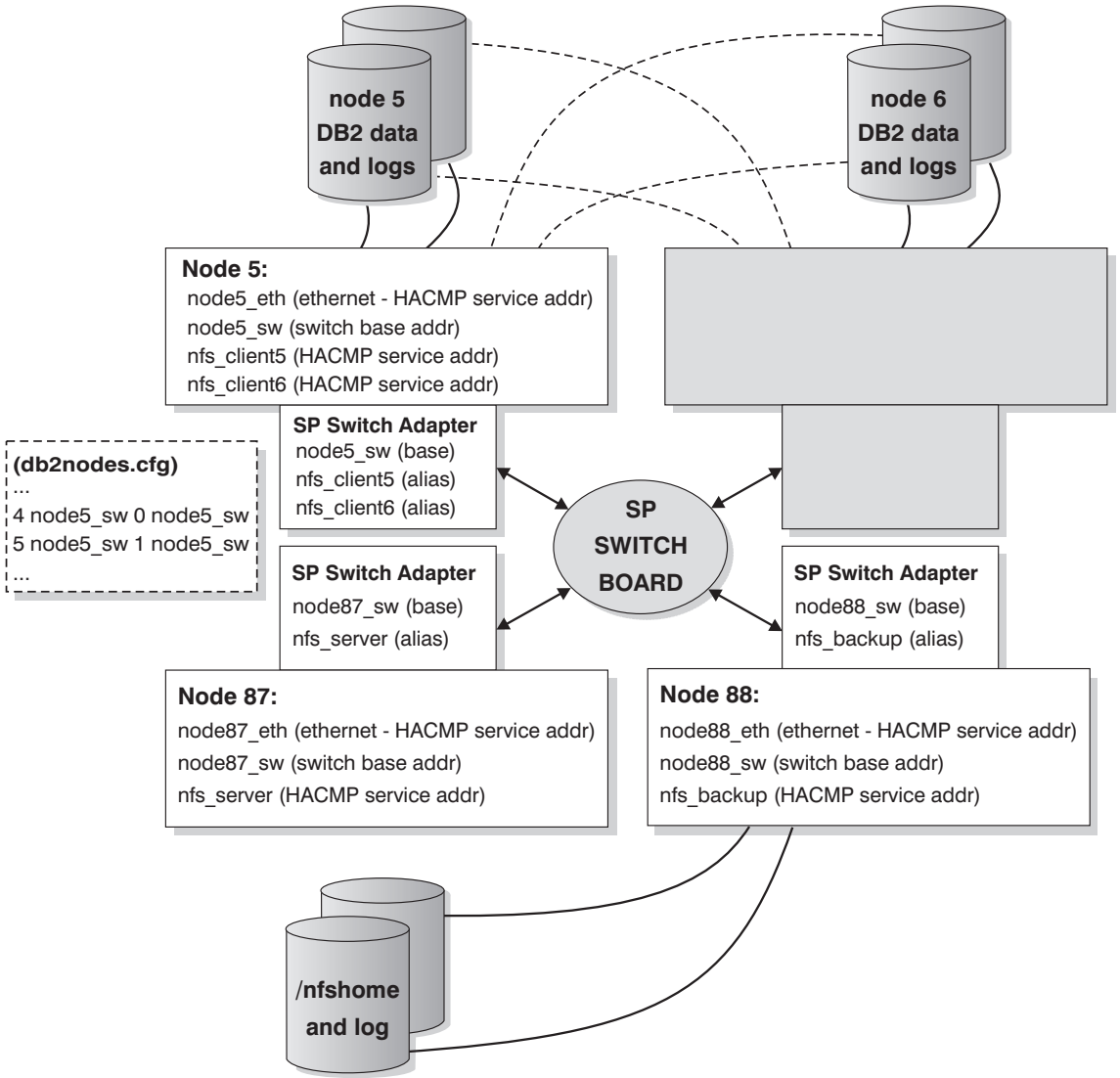


Figure 58. Mutual Takeover with NFS Failover - DB2 Failover

DB2 HACMP Hot Standby with NFS Failover - Normal

Note: Hot Standby node can back up more than one node, depending on disk cabling.

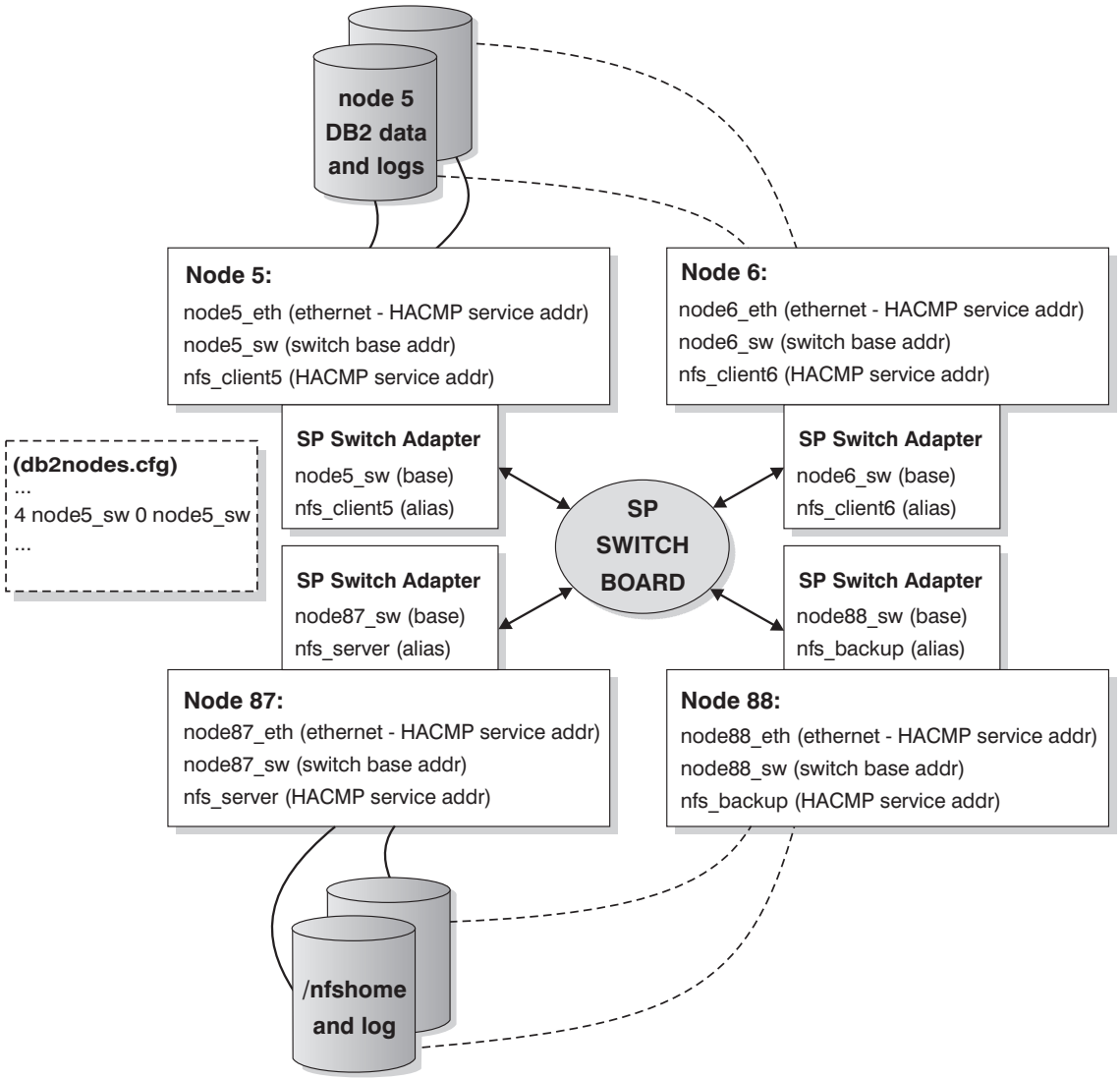


Figure 59. Hot Standby with NFS Failover - Normal

The previous figure and the next figure each have the following notes associated with them:

1. HACMP adapters are defined for ethernet, and SP Switch alias boot and service aliases — base addresses are untouched. Remember to use a “HPS” string in the HACMP network name.
2. The NFS_server/nfshome is mounted as /dbhome on all nodes through switch aliases.
3. The db2nodes.cfg file contains SP Switch base addresses. The db2nodes.cfg file is changed by the DB2START RESTART command after a DB2 database partition (logical node) failover.
4. The Switch alias boot addresses are not shown.

DB2 HACMP Hot Standby with NFS Failover- DB2 Failover

Note: Hot Standby node can back up more than one node, depending on disk cabling.

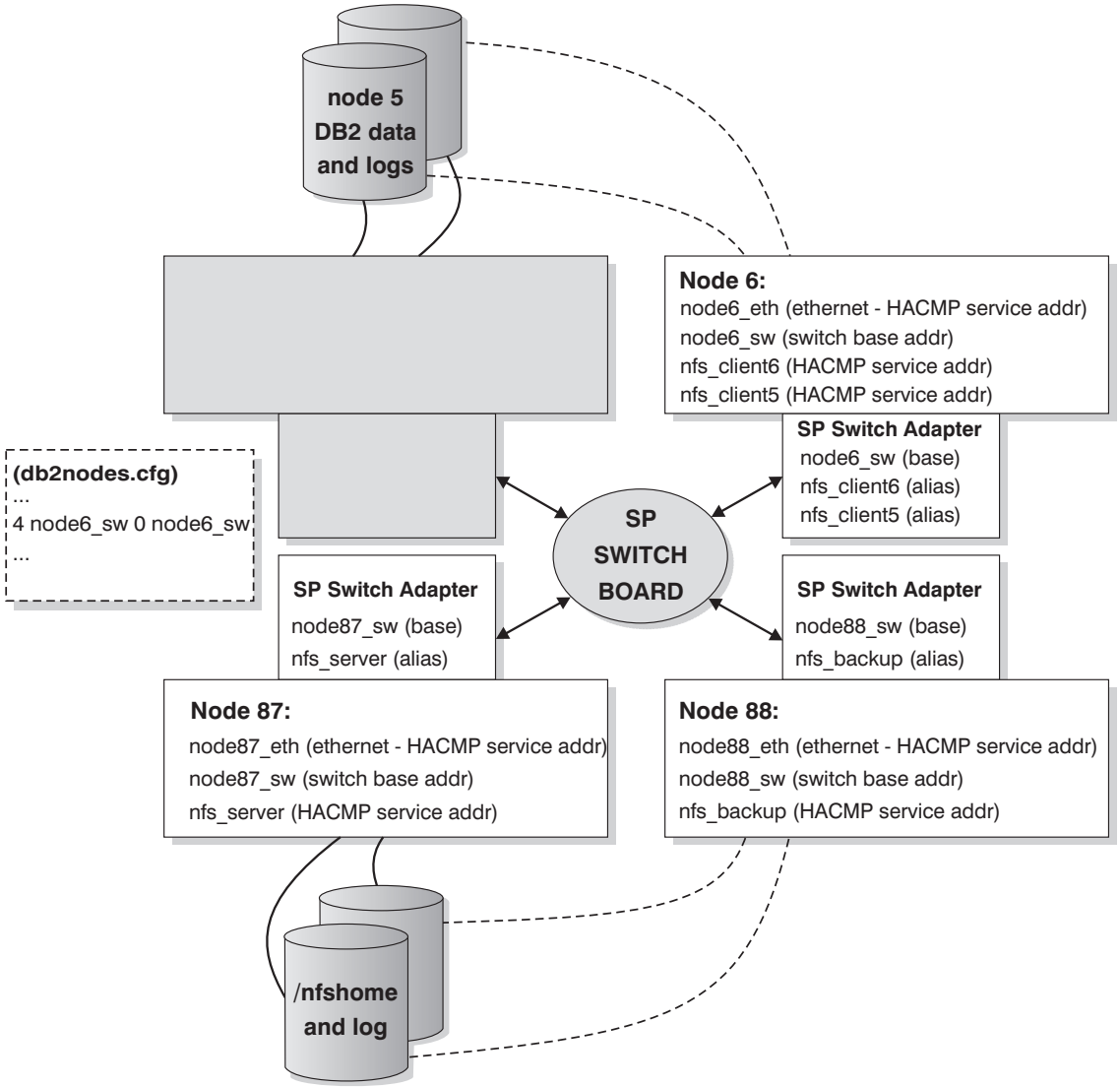


Figure 60. Hot Standby with NFS Failover - DB2 Failover

DB2 HACMP Mutual Takeover without NFS Failover - Normal

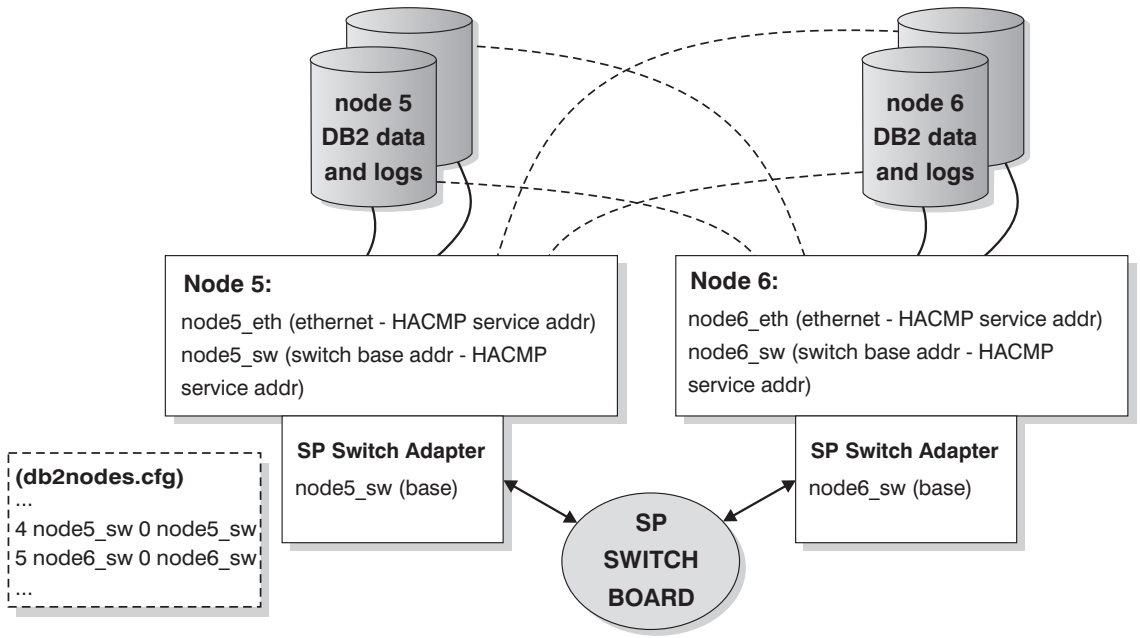


Figure 61. Mutual Takeover without NFS Failover - Normal

The previous figure and the next figure each have the following notes associated with them:

1. HACMP adapters are defined for ethernet, and SP Switch base addresses. Remember that when bases addresses are configured to HACMP as service addresses, there is no boot address (only a “heartbeat”).
2. Do not forget to use a “HPS” string in the HACMP network name for the SP Switch.
3. The `db2nodes.cfg` file contains SP Switch base addresses. The `db2nodes.cfg` file is changed by the DB2START RESTART command after a DB2 database partition (logical node) failover.
4. No NFS failover functions are shown.
5. Nodes can be in different SP frames.

DB2 HACMP Mutual Takeover without NFS Failover - DB2 failover

- Node 5 runs 2 logical nodes of DB2.

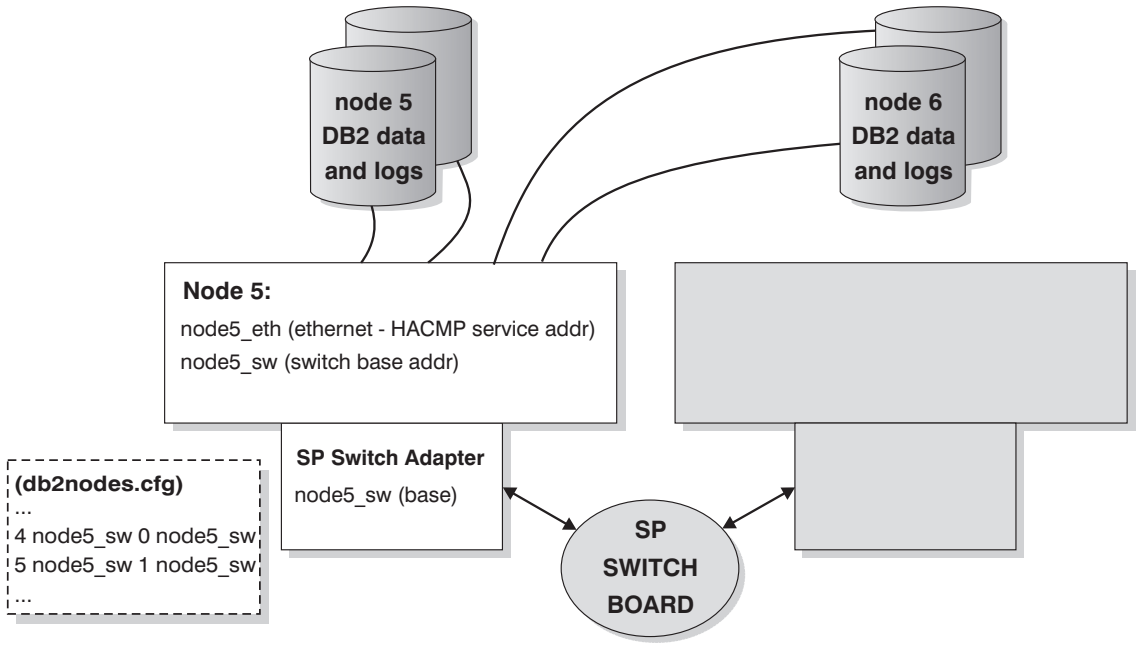


Figure 62. Mutual Takeover without NFS Failover - DB2 Failover

DB2 HACMP Startup Recommendations

It is recommended that you do not specify HACMP to be started at boot time in `/etc/inittab`. HACMP should be started manually after the nodes are booted. This allows for non-disruptive maintenance of a failed node.

As an example of “disruptive maintenance”, consider the case where a node has a hardware failure and crashed. At such a time, service needs to be performed. Failover would be automatically initiated by HACMP and recovery completed successfully. However, the failed node needs to be fixed. If HACMP was configured to be started on reboot in `/etc/inittab`, then this node would attempt to reintegrate after boot completion which is not desirable in this situation.

As an example of “non-disruptive maintenance”, consider manually starting HACMP on each node. This allows for non-disruptive service of failed nodes

since they can be fixed and reintegrated without affecting the other nodes. The ha_cmd script is provided for controlling HACMP start and stop commands from the control workstation.

HACMP ES Event Monitoring and User-Defined Events

The following is an example of a user-defined event: Perhaps you want to shut down DB2 database partitions on an AIX physical node when paging space reaches a certain percentage of fullness, and to log this occurrence. An example to correct a paging space shortage by shutting down a database partition and forcing a transaction abort to free paging space is provided. The examples are found in the /SAMPLES directory. Another common example is process death: You may want to restart a DB2 database partition, or you may want failover to occur if a process dies on a given node.

With HACMP ES there is a rules file, /user/sbin/cluster/events/rules.hacmprd, that contains HACMP events.

Each event in the file is made up of nine lines which are:

1. Event name. Each event name must be unique.
2. State. This is the qualifier for the event. The event name and state are the rule triggers. HACMP ES Cluster Manager initiates recovery only if it finds a rule with a trigger corresponding to the event name and state.
3. Resource Program Path. This is a full-path specification of the xxx.rp file containing the recovery program.
4. Recovery Type. This is reserved for future use.
5. Recovery Level. This is reserved for future use.
6. Resource Variable Name. This is used for Event Manager events.
7. Instance Vector. This is used for Event Manager events. Within Event Management, this is a set of elements, where each element is a name and value pair of the form “name=value”. The values uniquely identify the copy of the resource in the system and, by extension, the copy of the resource variable.
8. Predicate. This is used for Event Manager events. Within Event Management, this is the relational expression between a resource variable and other elements that, when true, the Event Management subsystem generates an event to notify Cluster Manager and the appropriate application.
9. Rearm Predicate. This is used for Event Manager events. Within Event Management, this is a predicate used to generate an event that alternates the status of the primary predicate. This predicate is typically the inverse of the primary predicate. It can also be used with the event predicate to establish an upper and a lower boundary for a condition of interest.

Each object requires one line in the event definition even if the line is not used. If these lines are removed, HACMP ES Cluster Manager cannot parse the event definition properly. And this may cause the system to hang. Any line beginning with "#" is treated as a comment line and is not treated as part of the event definition.

Note: The rules file requires exactly nine lines for each event definition not counting any comment lines. When adding a user-defined event at the bottom of the rules file, it is important to remove the unnecessary empty line at the end of the file, or the node will hang.

An example of an event definition for node_up follows:

```
##### Beginning of the Event Definition: node_up
#
TE_JOIN_NODE
0
/usr/sbin/cluster/events/node_up.rp
2
0
# 6) Resource variable - only used for event management events

# 7) Instance vector - only used for event management events

# 8) Predicate  - only used for event management events

# 9) Rearm predicate - only used for event management events

###### End of the Event Definition: node_up
```

This is an example of just one of the event definitions that are found in the rules.hacmprd file.

In this example, when the node_up event occurs, the recovery program /usr/sbin/cluster/events/node_up.rp is executed. According to the rules, the proper values are specified in the state, recovery type, and recovery level lines in the definition. There are four (4) empty lines for: resource variable, instance variable, predicate, and rearm predicate.

Users can add their own events to react to non-standard HACMP ES events. For example, to define the event that the /tmp file system is over 90 per cent full, the rules.hacmprd file must be modified.

Many events are predefined in the IBM Parallel System Support Program (PSSP). These events can be exploited when used within user-defined events. To make this happen, do the following:

1. Stop the cluster.
2. Edit the rules.hacmprd file. Backup the file before modifying it. Add the predefined PSSP event manually. If you need synchronizing points across

all nodes in the cluster, use the barrier command in the recovery program. (Read more about the barrier command and synchronization of recovery programs in the HACMP Concepts, Installation, and Administration Guides.)

3. Restart the cluster. The rules.hacmprd file is stored in memory when Cluster Manager is started. To accurately implement the changes, restart all the clusters. There should not be any inconsistent rules in a cluster.
4. Cluster Manager uses all events in the rules.hacmprd file.

HACMP ES uses PSSP event detection to treat user-defined events. The PSSP Event Management subsystem provides comprehensive event detection by monitoring various hardware and software resources.

Resource states are represented by resource variables. Resource conditions are represented as expressions called predicates.

Event Management receives resource variables from the Resource Monitor, which observes the state of specific system resources and transforms this state into several resource variables. These variables are periodically passed to Event Management. Event Management applies predicates that are specified by the HACMP ES Cluster Manager in rules.hacmprd to each resource variable. When the predicate is evaluated as being true, an event is generated and sent to the Cluster Manager. Cluster Manager initiates the voting protocol and the recovery program file (xxx.rp) is executed on a set of nodes specified by “node sets” in the recovery program and according to event priority.

The recovery program file (xxx.rp) is made up of one or more recovery program lines. Each line is declared in the following format:

```
relationship     command_to_run     expected_status     NULL
```

There must be at least one space between each value in the format. “Relationship” is a value used to decide which program should run on which kind of node. Three types of relationship are supported:

- All. The specified command or program is executed on all nodes of the current HACMP cluster.
- Event. The specified command or program is executed only on the nodes where the event occurred.
- Other. The specified command or program is executed on all nodes where the event did not occur.

“Command_to_run” is a quote-delimiting string with or without a full-path definition to an executable program. Only HACMP-delivered event scripts can use a relative-path definition. With other scripts or programs, the full-path definition must be used (even if these programs are located in the same directory as the HACMP event scripts). “Expected_states” is the return code

of the specified command or program. It is an integer value or an “x”. If “x” is used, Cluster Manager does not care about the return code. For all other codes, it must be equal to the expected return code. If it is not, Cluster Manager detects the event failure. The handling of this event “hangs” the process until the problem is solved through a manual intervention to recover. Without manual intervention, the node does not hit the barrier to synchronize with the other nodes. Synchronization across all nodes is a requirement for the Cluster Manager to control all the nodes. “NULL” is a field reserved for future use. The word “NULL” must appear at the end of each line except the barrier line. If you specify multiple recovery commands between two barrier commands, or before the first one, the recovery commands are executed in parallel on the node itself and between the nodes.

The barrier command is used to synchronize all the commands across all the cluster nodes. When a node hits the barrier statement in the recovery program, Cluster Manager initiates the barrier protocol on this node. Since the barrier protocol is a two-phase protocol, when all nodes have met the barrier in the recovery program and “voted” to approve the protocol, then all nodes are notified that both phases have completed.

In summary, the following actions make up the process:

1. Either Group Services/ES for predefined events, or Event Management for user-defined events, notifies Cluster Manager of the event.
2. HACMP ES Cluster Manager reads the `rules.hacmprd` file and determines the recovery program mapped to the event.
3. HACMP ES Cluster Manager runs the recovery program which consists of a sequence of recovery commands.
4. The recovery program executes the recovery commands which may be shell scripts or binary commands.

   Note: The recovery commands are the same as the HACMP event scripts in HACMP for AIX.
5. HACMP ES Cluster Manager receives the return status from the recovery commands. An unexpected status “hangs” the cluster until manual intervention using `smit cm_rec_aids` or the `/usr/sbin/cluster/utilities/clruncmd` command is carried out.

HACMP ES Script Files

Included with DB2 UDB EEE are sample scripts for failover/recovery and for user-defined events. The scripts will work “as is” or you can customize or change the recovery action.

- DB2 database partition recovery script rc.db2pe. This is the script file used to start and stop the HACMP configuration on a database partition. It also works as a HACMP start and stop script for a NFS server of the DB2 instance owner.
- DB2-specific user-defined events for HACMP ES. Six default events are included: one for process recovery, two for paging space, and three for NFS and automounter recovery.
- DB2 instance NFS fileserver failover. This script provides for failover recovery of the server of the filesystem for a DB2 instance to a backup.
- Network failover. The scripts network_up_complete, network_back and network_down_complete, network_down allow SP DB2 database partitions to failover if their SP Switch adapter should fail.
- Scripts to define monitoring events for the SP GUI Perspectives are included. Monitoring of failover and user-defined recovery is possible through the Event and Hardware Perspectives. Read the documentation for PSSP Administration to find out more about Perspectives.
- Installation scripts to install and remove core scripts and events on the HACMP ES nodes.
- Script files to create and remove the SP Perspectives problem management (pman) resources for monitoring the HACMP and DB2 configuration.

The script files are located in the DB2 UDB EEE $INSTNAME/sqllib/samples/hacmp/es directory.

The recovery scripts need to be installed on each node that will run recovery. The script files can be centrally installed from the SP control workstation or other designated SP node. To install, complete the following tasks:

1. Copy the scripts from the $INSTNAME/sqllib/samples/hacmp/es directory to one of either the SP control workstation or another SP node that can run the pcp and pexec commands. (The pcp and pexec commands are required for the install so ensure that you have the ability to run them.)
2. Customize the reg.parms.SAMPLE and failover.parms.SAMPLE files for your environment by setting key parameters such as BUFFPAGE for failover configurations. Typically for mutual takeover configurations, your failure settings will be adjusted lower to one-half the size of your regular settings or less. Also, you will use a copy of these files renamed with your own name (instead of “SAMPLE”).
3. Customize as necessary the five (5) parameters NFS_RETRIES, START_RETRIES, MOUNT_NFS, STOP_RETRIES, and FAILOVER in the rc.db2pe file. The three retries and the single failover settings should be adequate for almost all implementations. The MOUNT_NFS setting should be configured depending on whether you will be using the package for NFS server availability. You should specify this setting if you wish rc.db2pe to mount and verify the NFS home directory of the DB2 instance

owner for you. Setting the FAILOVER parameter to "YES" will cause the running of db2_proc_restart and attempt to restart a DB2 database partition. If unsuccessful in this attempt, HACMP will be shutdown with a failover.

4. Customize db2_paging_action, db2_proc_recovery, and nfs_auto_recovery in the event file. Also, edit pwq to change this to the DB2 instance owner. Customize the db2_paging_action to indicate the action to take if paging space gets more that ninety percent full. (If this does occur, the DB2 database partition is stopped.) Modify the script if additional recovery actions are required.
5. Use db2_inst_ha to install the scripts and events on the nodes you specify.

   Note: HACMP ES must be pre-installed on these nodes before you begin. The syntax of db2_inst_ha is:

   ```
   db2_inst_ha $INSTNAME/sqllib/samples/hacmp/es <nodelist> <DATABASENAME>
   ```

   where

   $INSTNAME/sqllib/samples/hacmp/es is the directory where the scripts/event are located
   <nodelist> is the pcp or pexec style of nodes; for example, 1-16 or 1,2,3,4
   <DATABASENAME> is the name of the database for regular and failover parameter files.

   The reg.parms.SAMPLE and failover.parms.SAMPLE will be copied to each node and renamed reg.parms.DATABASENAME. db2_inst_ha will copy files to each node in /usr/bin and update the HACMP event files: /usr/sbin/cluster/events/rules.hacmprd, /usr/sbin/cluster/events/network_up_complete, and /usr/sbin/cluster/events/network_down_complete.
6. Configure your system and scripts with HACMP.
7. Use the create_db2_events command to install the monitoring events for problem management resources (pman) and the SP GUI Perspectives. Additional configuration and customization in Perspectives is needed. For more information on Perspectives, read the PSSP Administration Guide.
8. Use the ha_db2stop command to shutdown the database partitions without HACMP ES failover recovery taking place. To use this command, copy the file to the database user's home directory and make sure permissions and ownership are set for that user. To stop the database without failover recovery, then as that user, type:

   ```
   ha_db2stop
   ```

Note: You must wait for the command to return. Exiting by using a ctrl-C interrupt, or by killing the process, may re-enable failover recovery prematurely. This would result in not all database partitions being stopped.

DB2 Recovery Scripts Operations with HACMP ES

HACMP ES invokes the DB2 recovery scripts in the following way:

- `node_up_local` (starting a node)
  1. HACMP will run the `node_up` sequence, acquiring volume groups, logical volumes, filesystems, and IP addresses specified in resource groups owned (via cascading) or assigned (via rotating) to this node.
  2. When `node_up_local_complete` is run, the application server definition which contains `rc.db2pe` is initiated to start the database partition specified in the application server definitions on this physical node.

     Note: `rc.db2pe`, when running in start mode, adjusts the DB2 parameters specified in `reg.parms.DATABASE` for each DATABASE in the database directory that matches a parameter (parms) file.

  Each node, when starting, follows this sequence. If you have multiple HACMP clusters and start them in parallel, multiple nodes are brought up at once.
- `node_down_remote` (failover)
  1. HACMP will acquire volume groups, logical volumes, filesystems, and IP addresses specified in the resource group on the designated takeover node.
  2. When `node_down_remote_complete` is run, HACMP will run `rc.db2pe` as the application server start script specified in the resource group for this database partition.

     Note: `rc.db2pe`, when running in a takeover mode (mutual takeover), will stop the DB2 database partition running on it, adjust the DB2 parameters specified in `failover.parms.DATABASE` for each DATABASE in the database directory that matches a parameter (parms) file, and then starts both database partitions on the physical takeover node.
- `node_up_remote` (reintegration of a failed node - cascading mutual takeover resource group)
  1. When `node_up_remote` is run on the old takeover node, the application server definition causes `rc.db2pe` to be run in stop mode.

     Note: `rc.db2pe`, when running in a reintegration mode (mutual takeover), will stop both of the database partitions running on it, adjust the DB2 parameters specified in `reg.parms.DATABASE` for

each DATABASE in the database directory that matches a parameter (parms) file, and then starts just the database partition to be kept on this physical takeover node.

2. The old takeover node releases volume groups, logical volumes, filesystems, and IP addresses specified in resource groups to be owned by the reintegrating node.
3. HACMP will re-acquire volume groups, logical volumes, filesystems, and IP addresses specified in the resource group now owned by the reintegrating node.
4. When node_up_local_complete is run, the application server definition which contains rc.db2pe is initiated to start the DB2 database partition specified in the application server definition on this reintegrating physical node.

   Note: rc.db2pe, when running in start mode will adjust the DB2 parameters specified in reg.parms.DATABASE for each DATABASE in the database directory that matches a parameter (parms) file.

- node_down_local (node stop or stop with takeover)
  1. When node_down_local is run on the stopping node, the application server definition causes rc.db2pe to be run in stop mode.

     Note: rc.db2pe, when running in a stop mode will adjust the DB2 parameters specified in failover.parms.DATABASE for each DATABASE in the database directory that matches a parameter (parms) file, and then stops the DB2 database partition (this is for takeover).
  2. HACMP releases volume groups, logical volumes, filesystems, and IP addresses specified in resource groups now owned by the node.
- db2_proc_recovery (db2 process death)
  1. All nodes run the db2_proc_restart script. The node which had the failure restarts the correct DB2 database partition.
- db2_paging_recovery (paging space recovery)
  1. All nodes run the db2_paging_action script. If a node has more than seventy (70) percent of paging space filled, a wall command is issued. If a node has more than ninety (90) percent of paging space filled, then DB2 database partitions on this physical node are stopped and restarted.
- nfs_auto_recovery (nfs or automount process failure)
  1. All nodes run the rc.db2pe script in NFS mode. If a NFS process stops running, then it is restarted. In a similar way, if the automount process stops running then it is restarted.
- network_down_complete (network failure - SP switch)

1. The net_down script is called. This verifies the network as the SP switch network and verifies it is down. If so, it waits a user-defined time interval. The default time interval is one hundred (100) seconds.
2. If the SP switch network comes back as indicated by network_up_complete event, then no recovery is effected.
3. If the time limit is reached, then HACMP is stopped with failover.

Note: All events can be monitored through SP problem management and the SP Perspectives GUI.

Other Script Utilities

There are other script utilities available for your use which include:

- ha_cmd is a command provided to start HACMP on SP nodes from the control workstation. The syntax of the command is:

  ```
  ha_cmd <noderange> <START|STOP|TAKE|FORCE>
  ```

  where <noderange> is a pcp or pexec style of SP noderange. For example, ha_cmd 3-6 START would start HACMP on nodes 3,4,5,6. ha_cmd 5 TAKE would shutdown HACMP on node 5 for takeover.
- ha_mon is a command for monitoring HACMP hacmp_out files from the SP control workstation. To invoke this command, type ha_mon <node> where <node> is the SP node to be monitored. ha_mon will “tail -f” the /tmp/hacmp.out file on the node you specify.
- db2_turnoff_recov is a command designed for extremely rare situations. This command temporarily disables all HACMP (non-failover) recovery. No DB2 process, paging, NFS, or automounter recovery is initiated. This function removes the event stanzas for that recovery from the HACMP rules file. HACMP must be stopped and restarted. To invoke this command, type db2_turnoff_recov <nodelist>.
- db2_turnon_recov is a command to re-enable HACMP (non-failover) recovery. This command would be used after db2_turnoff_recov to restore HACMP rules files so that user-defined event recovery can occur. HACMP must be stopped and restarted. To invoke this command, type db2_turnon_recov <nodelist>.

Monitoring HACMP Clusters

There are scripts provided for creating SP problem management (pman) events to monitor the DB2 HACMP ES configuration, in addition to those monitoring utilities already present in HACMP ES. To monitor HACMP status form the SP control workstation, do the following:

- Install the HACMP client code on the control workstation.

- Edit the /usr/sbin/cluster/etc/clhosts file and include the SP ethernet IP addresses of the nodes you wish to monitor.
- Use the command startsrc -s clinfo to start monitoring the clusters.

HACMP supplies an interface for monitoring the clusters: /usr/sbin/cluster/clstat.

To use the problem management monitoring with SP Perspectives GUI for HACMP RS and user-defined events:

1. Use create_db2_events <nodelist> where <nodelist> is a pcp or pexec style of nodes and where the events are to be monitored. create_db2_events creates five (5) pman events for monitoring by Perspectives.

   Note: The Resource Variables PSSP.pm.User_state12-16 are used in the creation of these events. If these resource variables are already being used for some other purpose, create_db2_events and update_db2_events must be updated to use different resource variables.
2. Start Perspectives on the control workstation. From the launch pad, choose the event Perspective. You should see five (5) events: db2_hacmp_recovery, db2_process_recovery, db2_paging_err, db2_nfs_err, and Errlog_PERM_entry.
3. Double-click on each event. On the screen that appears, you need to register (within the Definition Table) a condition for the event. Click next to the down arrow by Name: "unnamed", and select the same name as the event you specify as the condition. Select the "Response Options" tab. Click on the button on the top of the display ("Send Message to Perspectives event session"). If you desire, you can specify commands, Errlog entries, as well as SNMP traps for these event occurrences. The event log displays are maintained only across Perspective sessions; therefore, you might want to create AIX error log entries for each. Hit the "OK" Button, and close the window.
4. Next, go back to the Perspectives launch pad. Select the hardware Perspective.
5. When the Hardware frame GUI appears, select at the top of the menu "View" and then "Monitor". You are then provided with a list of events that can be monitored for your SP. Scrolling to the bottom of the list, you will find two additional events: one for HACMP DB2 recovery (db2_ha_ind), and the other for SP node PERM errors (Errlog_PERM_mon. Select those you wish to monitor. (When an event occurs for a node, it will receive a red "X" in its display. If all monitored conditions are "OK", the display for the node is green.) Typically, host_responds, switch_responds,

and node_power_LED are used. You can also monitor the DB2 HACMP recovery as well as PERM errors on the node.

Note: The db2_hacmp_mon and db2_hacmp_recovery variables for pman and Perspectives do not reflect HACMP cluster status. Rather, these variables reflect the status of the rc.db2pe operation to start or stop DB2. The "real" HACMP status is shown in the HACMP clstat monitor and reflects the HACMP cluster state. If you wish db2_hacmp_ind to reflect monitoring similar to HACMP Status, add the following line to your /etc/inittab file:

```
haind:2:wait:/usr/bin/db2_update_events HAIND OFF 2>&1 >/dev/null
```

If you are planning on using NetView for your implementation, consider using HAVIEW (which is part of HACMP) for monitoring your configuration. Please use NetView documentation for information on configuring that product.

DB2 SP HACMP ES Installation

To assist in the planning for the installation of HACMP ES on DB2 UDB, a step-by-step overview of the installation and migration processes is presented here.

DB2 SP HACMP ES New Installation

When planning for and implementing HACMP ES in an environment where you have not installed HACMP before, you should consider the following tasks:

1. Install the AIX operating system on each of the SP nodes according to the SP Installation and Administration Guides. Ensure proper paging space is available on both the control workstation and each of the SP nodes. Also ensure switch configuration has been considered and implemented along with any other modifiable configuration parameters. In addition, SP monitoring (Perspectives) you desire to use should be put in place. Ensure the SP dsh, pcp, and pexec commands work.
2. Design your database layout. This should, at a minimum, include the number of nodes to be used, the mapping of DB2 database partitions to physical nodes, the disk requirements per node/partition, and table space considerations. You should also consider who the main DB2 instance owner will be and the access authorization this and other users will require.
3. Plan your external SSA disk configuration including redundant adapters, mirrored disks, and the twin-tailing of disks.

4. Using your database layout and SSA configuration, complete the HACMP worksheets found in the HACMP Planning, Installation, and Administration Guides. Using these worksheets, you should be able to complete the worksheets later in this document.
5. Implement your external SSA disk configuration. Make sure microcode levels are consistent across all drives and use the Maymap utility for validating and filling in any gaps in your worksheets.
6. Install DB2 UDB EEE on each SP node.
7. Install HACMP ES on each SP node.
8. Install the DB2 UDB EEE HACMP ES on SP Package using the `db2_inst_ha` command.
9. Create the DB2 main instance user and validate it can access all nodes. This is not a highly available user at this point. This can be temporarily a SP user on the SP control workstation.
10. Create your DB2 instance and database. Ensure it is operating by using `db2start` command. Then ensure it is stopped by using `db2stop` before proceeding to the next step.
11. If you wish to implement or load the database before adding HACMP, then you should do this now.
12. Configure HACMP ES on the SP nodes topology and resource groups according to the HACMP worksheets and the information in this document.
13. Beginning with your NFS server node for the DB2 main instance user, change this user (by modifying `/etc/security/user` and `/etc/passwd` on all nodes in accordance with what is specified in this document. This user will become a highly available NFS user; and this node and its backup will update `/etc/exports`. All nodes will be able to mount this directory using NFS (with an entry in `/etc/filesystems` on each node) through the switch alias IP addresses.
14. "Tar" the home directory of the main instance user and "un-tar" the home directory in the new location.
15. Create a NFS filesystem on each of the SP nodes to mount a new main instance home directory.
16. Start HACMP on the NFS server node. Verify that it comes up successfully by investigating `/tmp/hacmp.out`. The `ha_mon` command can be used to monitor this file as it is written.
17. Bring up the other nodes one at a time; verifying each successful completion by investigating `/tmp/hacmp.out`. The `ha_mon` command can be used to monitor this file as it is written.
18. Setup the optional monitoring through Perspectives and Problem Management.

19. Validate failover functionality on each node by simulating a concurrent maintenance action on each node. The `ha_cmd nodenum TAKE` can be used to stop HACMP gracefully with takeover. Verify the takeovers and reintegrations succeed by interrogation of `/tmp/hacmp.out` and your monitoring tools.

DB2 SP HACMP ES Migration

If you are migrating from a non-HACMP installation to one with HACMP, you should review the step-by-step overview that follows:

1. Convert your existing external disks to a highly-available, twin-tailed, mirrored configuration. Add any extra hardware and disks to achieve this configuration remembering that names of different logical volumes on different nodes must be unique when they are twin-tailed. This applies to volume groups, logical volumes, and filesystems.
2. Complete the HACMP planning and the related worksheets. Also, complete the worksheets in this document.
3. Implement your external SSA disk configuration changes. Ensure microcode levels are consistent across all drives and use the Maymap utility to validate and eliminate any gaps in the worksheets.

   Note: SSA disks in a RAID5 configuration is supported. Two SSA adapters in the same RAID loop is the only configuration permitted. For a HACMP configuration with the RAID disks twin-tailed, only one adapter per node is supported. In this configuration, the adapter is a single point of failure for access to the disks, and extra configuration is recommended to detect the adapter outage adn promote this to an HACMP failover event. AIX error notification is the simplest way to configure a node for fail over should the SSA adapter fail. Refer to *HACMP for AIX, V4.2.2, Enhanced Scalability Installation and Administration Guide* for more information on AIX error notification.
4. Install HACMP ES on each SP node.
5. Install the “DB2 UDB EEE HACMP ES on SP” Package using the `db2_inst_ha` command.
6. Configure HACMP ES on the SP nodes topology and resource groups according to the HACMP worksheets and the information in this document.
7. Beginning with your NFS server node for the DB2 main instance user, change this user (by modifying `/etc/security/user` and `/etc/passwd` on all nodes in accordance with what is specified in this document. This user will become a highly available NFS user; and this node and its backup will update `/etc/exports`. All nodes will be able to mount this directory using NFS (with an entry in `/etc/filesystems` on each node) through the switch alias IP addresses.

8. “Tar” the home directory of the main instance user and “un-tar” the home directory in the new location.
9. Create a NFS filesystem on each of the SP nodes to mount a new main instance home directory.
10. Start HACMP on the NFS server node. Verify that it comes up successfully by investigating `/tmp/hacmp.out`. The `ha_mon` command can be used to monitor this file as it is written.
11. Bring up the other nodes one at a time; verifying each successful completion by investigating `/tmp/hacmp.out`. The `ha_mon` command can be used to monitor this file as it is written.
12. Setup the optional monitoring through Perspectives and Problem Management.
13. Validate failover functionality on each node by simulating a concurrent maintenance action on each node. The `ha_cmd nodenum TAKE` can be used to stop HACMP gracefully with takeover. Verify the takeovers and reintegrations succeed by interrogation of `/tmp/hacmp.out` and your monitoring tools.

DB2 SP HACMP ES Worksheets

The worksheets below are designed to be used with the HACMP worksheets that were filled out in preparation for your configuration.

In each of two cases, first a worksheet is filled out to give you an idea of how to plan your configuration. Secondly, a blank sample worksheet is provided for your use.

The database configuration on external disks documented in the first sample worksheet is shown in the following figure. The database statement used to create the database was:

```
db2 create database pwq on /newdata
```

Both SSA external adapters and external SSA disks are mirrored and twin-tailed for logical volumes with no single point of failure. The diagram pictured is quite similar to the output of the `maymap` command. Maymap is a utility available through AIXTOOLS to show the external SSA disk configuration. Use of this utility is recommended as part of planning your setup.

Sample DB2 4-node Database External Disks Setup

- Showing twin-tailing for High Availability.

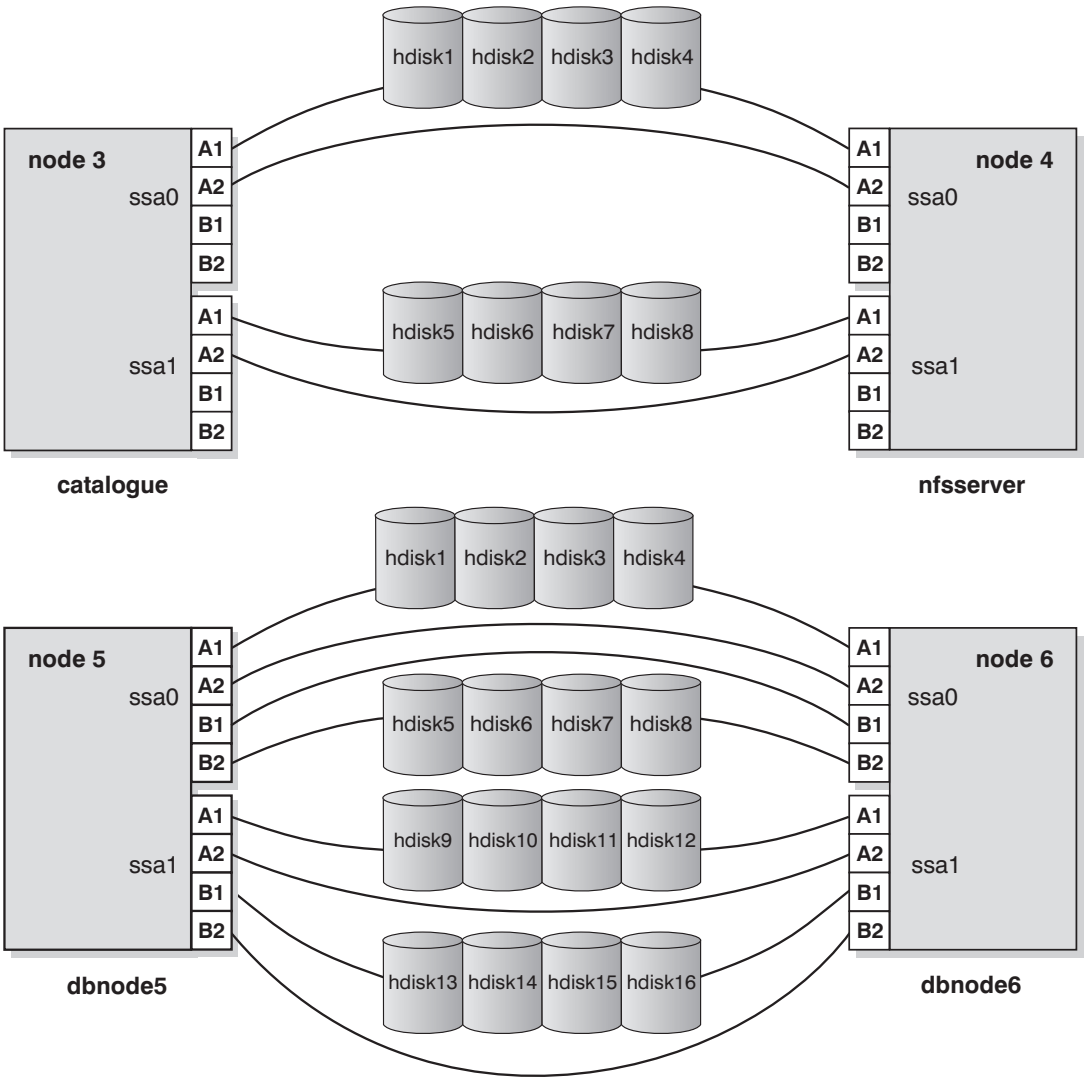


Figure 63. Sample DB2 4-node Database External Disks Setup

Before you review the following table, you are expected to have thoroughly read the HACMP documentation regarding the quorum settings on volume groups and mirrored write consistency settings on logical volumes. The settings used for both will directly affect your availability and performance.

Ensure you review these settings and understand their implications. The typical setting for both “quorum” and “mirrored write consistency” is “off”.

Table 36. HACMP Volume Groups, Logical Volumes, and Filesystems

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Cop -ies	hdisk list	Filesystem Mount Point (MB)	Filesystem Log logical volume	Node Description and backup	user owner of /dev logical device
3	havg3	8	hlv300	10	2	hdisk1 hdisk5	/newdata /pwq /NODE0003	hlog301	Catalognode mount point; node 4	root *
3	havg3	8	hlog301	1	2	hdisk1 hdisk5	N/A	N/A	Catalognode jfslog; node 4	root *
3	havg3	8	hlv301	10	2	hdisk2 hdisk6	N/A	N/A	Catalognode rawtemp space; node 4	pwq **
4	havg4	8	hlv400	10	2	hdisk3 hdisk7	/dbmnt	hlog401	nfsserver pwq home; node 3	root *
4	havg4	8	hlog401	1	2	hdisk3 hdisk7	N/A	N/A	nfsserver jfslog; node 3	root *
5	havg5	8	hlv500	10	2	hdisk1 hdisk9	/newdata/ pwq/ NODE0005	HLOG501	Dbnode5 mount point; node 6	root *
5	havg5	8	hlog501	1	2	hdisk1 hdisk9	N/A	N/A	Dbnode5 jfslog; node 6	root *
5	havg5	8	hlv501	10	2	hdisk2 hdisk10	N/A	N/A	Dbnode5 raw temp space; node 6	pwq **
5	havg5	8	hlv502	100	2	hdisk2 hdisk10	N/A	N/A	Dbnode5 raw table space; node 6	pwq **
5	havg5	8	halv503	100	2	hdisk3 hdisk11	N/A	N/A	Dbnode5 raw table space; node 6	pwq **
5	havg5	8	halv504	100	2	hdisk3 hdisk11	N/A	N/A	Dbnode5 raw table space; node 6	pwq **
5	havg5	8	halv505	100	2	hdisk4 hdisk12	/dbdata5	hlog501	Dbnode6 system table space; node 6	root *
6	havg6	8	hlv600	10	2	hdisk5 hdisk13	/newdata/ pwq/ NODE0006	hlog601	Dbnode6 mount point; node 5	root *
6	havg6	8	hlog601	1	2	hdisk5 hdisk13	N/A	N/A	Dbnode6 jfslog; node 5	root *
6	havg6	8	hlv601	10	2	hdisk6 hdisk14	N/A	N/A	Dbnode6 raw temp space; node 5	pwq **
6	havg6	8	hlv602	100	2	hdisk6 hdisk14	N/A	N/A	Dbnode6 raw table space; node 5	pwq **

Table 36. HACMP Volume Groups, Logical Volumes, and Filesystems (continued)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Cop -ies	hdisk list	Filesystem Mount Point (MB)	Filesystem Log logical volume	Node Description and backup	user owner of /dev logical device
6	havg6	8	hlv603	100	2	hdisk7 hdisk15	N/A	N/A	Dbnode6 raw table space; node 5	pwq **
6	havg6	8	hlv604	100	2	hdisk7 hdisk15	N/A	N/A	Dbnode6 raw table space; node 5	pwq **
6	havg6	8	hlv605	100	2	hdisk8 hdisk16	/dbdata6	hlog601	Dbnode6 system table space; node 5	root *

Notes:

1. * jfs filesystem logical volumes and logs keep root permissions.
2. ** raw database spaces get database user permissions on /dev raw file entries (/dev/rxxxx).

Table 37. HACMP Volume Groups, Logical Volumes, and Filesystems (blank)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Cop -ies	hdisk list	Filesystem Mount Point (MB)	Filesystem Log logical volume	Node Description and backup	user owner of /dev logical device

Table 37. HACMP Volume Groups, Logical Volumes, and Filesystems (blank) (continued)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Cop -ies	hdisk list	Filesystem Mount Point (MB)	Filesystem Log logical volume	Node Description and backup	user owner of /dev logical device

Table 37. HACMP Volume Groups, Logical Volumes, and Filesystems (blank) (continued)

SP Node	Volume Group Name	PP Size (MB)	Logical Volume Name	# of PPs	Cop -ies	hdisk list	Filesystem Mount Point (MB)	Filesystem Log logical volume	Node Description and backup	user owner of /dev logical device

Table 38. Planning HACMP NFS Server

SP Node	External Filesystem	Back up node	SP switch boot and service IP alias pairs	filesystem to mount (/etc/ filesystems)	filesystem to specify as database home directory	addresses to export filesystem to (/etc/ exports)
3	/dbmnt	4	nfs_boot_3 nfs_client_3	nfs_server:/ dbmnt as /dbi	/dbi/pwq	nfs_boot_3 nfs_client_3 nfs_server_boot nfs_server nfs_boot_5 nfs_client_5 nfs_boot_6 nfs_client_6
4	/dbmnt	3	nfs_server_boot nfs_server	nfs_server:/ dbmnt as /dbi	/dbi/pwq	nfs_boot_3 nfs_client_3 nfs_server_boot nfs_server nfs_boot_5 nfs_client_5 nfs_boot_6 nfs_client_6
5	N/A	N/A	nfs_boot_5 nfs_client_5	nfs_server:/ dbmnt as /dbi	/dbi/pwq	N/A
6	N/A	N/A	nfs_boot_6 nfs_client_6	nfs_server:/ dbmnt as /dbi	/dbi/pwq	N/A

Notes:

1. `/etc/passwd` must be the same on all nodes. This can be synchronized from the control workstation.
2. Ensure the external filesystem has the permission of the database instance owner.
3. The `/etc/filesystems` must have the mount parameters: hard, bg, intr, and rw.
4. The `/etc/exports` will have

   ```
   -root=ip1:ip2:ip3
   ```

   only on the server and its backup.

Table 39. Planning HACMP NFS Server (blank)

SP Node	External Filesystem	Back up node	SP switch boot and service IP alias pairs	filesystem to mount (/etc/ filesystems)	filesystem to specify as database home directory	addresses to export filesystem to (/etc/ exports)

Table 39. Planning HACMP NFS Server (blank) (continued)

SP Node	External Filesystem	Back up node	SP switch boot and service IP alias pairs	filesystem to mount (/etc/ filesystems)	filesystem to specify as database home directory	addresses to export filesystem to (/etc/ exports)

Chapter 14. High Availability in the Windows NT Environment

You can set up your database system so that if a machine fails, the database server on the failed machine can run on another machine. On Windows NT, you implement failover support with Microsoft Cluster Server (MSCS). To use MSCS, you require Windows NT Version 4.0 Enterprise Edition with the MSCS feature installed.

MSCS can perform both failure detection and the restarting of resources in a clustered environment, such as failover support for physical disks and IP addresses. (When the failed machine is online again, resources will not automatically fall back to it, unless you previously configure them to do so. For more information, see "Fallback Considerations" on page 577.)

Before you enable DB2 instances for failover support, perform the following planning steps:

1. Decide which disks you want to use for data storage. Each database server should be assigned at least one disk for its own use. The disk that you use to store data must be attached to a shared disk subsystem, and must be configured as an MSCS disk resource.
2. Ensure that you have one IP address for each database server that you want to use to support remote requests.

When you set up failover support, it can be for an existing instance, or you can create a new instance when you implement the failover support.

To enable failover support, perform the following steps:

1. Create an input file for the DB2MSCS utility.
2. Run the DB2MSCS utility.
3. If you are using a partitioned database system, register database drive mapping to enable mutual takeover. See "Registering Database Drive Mapping for Mutual Takeover Configurations in a Partitioned Database Environment" on page 577.

After you finish enabling the instance for failover support, your configuration will resemble Figure 64 on page 566.

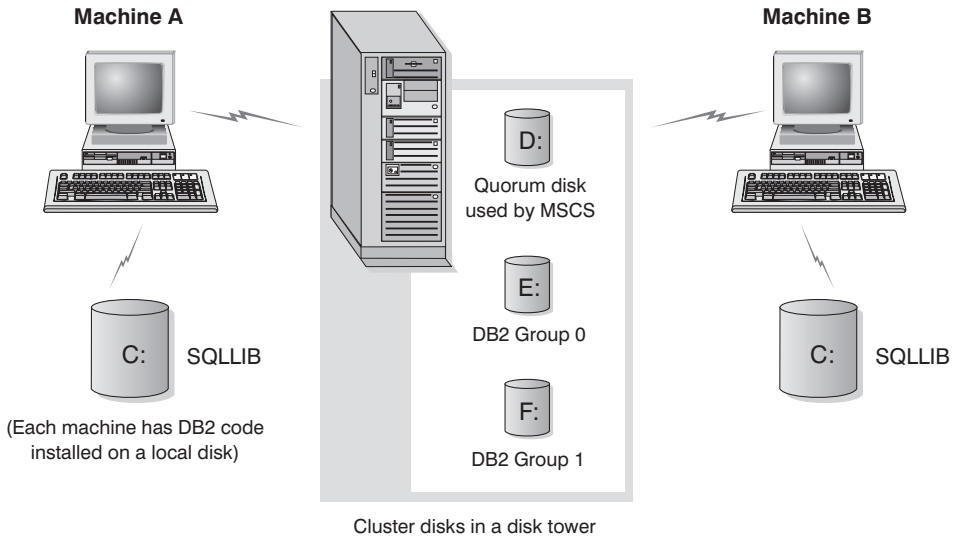


Figure 64. Example MSCS Configuration

The following sections describe the different types of failover support, and how to implement them. Before performing any of the steps described below, you must already have the MSCS software installed on every machine that you want to use in an MSCS cluster. In addition, you must also have DB2 installed on every machine.

Failover Configurations

Two types of configuration are available:

- Hot standby
- Mutual takeover

Currently, MSCS supports clusters of two machines.

In a partitioned database environment, the clusters do not all have to have the same type of configuration. You can have some clusters that are set up to use hot standby, and others that are set up for mutual takeover. For example, if your DB2 instance consists of five workstations, you can have two machines set up to use mutual takeover configuration, two to use hot standby configuration, with the last machine not configured for failover support.

Hot Standby Configuration

In a hot standby configuration, one machine in the MSCS cluster provides dedicated failover support, and the other machine participates in the database

system. If the machine participating in the database system fails, the database server on it will be started on the failover machine. If, in a partitioned database system, you are running multiple logical nodes on a machine and it fails, the logical nodes will be started on the failover machine. Figure 65 shows an example of a hot standby configuration.

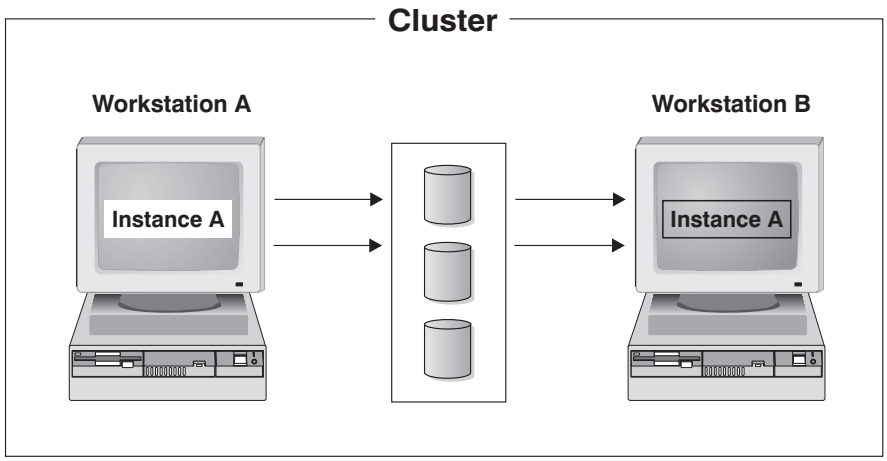


Figure 65. Hot Standby Configuration

Mutual Takeover Configuration

In a mutual takeover configuration, both workstations participate in the database system (that is, each machine has at least one database server running on it). If one of the workstations in the MSCS cluster fails, the database server on the failing machine will be started to run on the other machine. In a mutual takeover configuration, a database server on one machine can fail independently of the database server on another machine. Any database server can be active on any machine at any given point in time. Figure 66 on page 568 shows an example of a mutual takeover configuration.

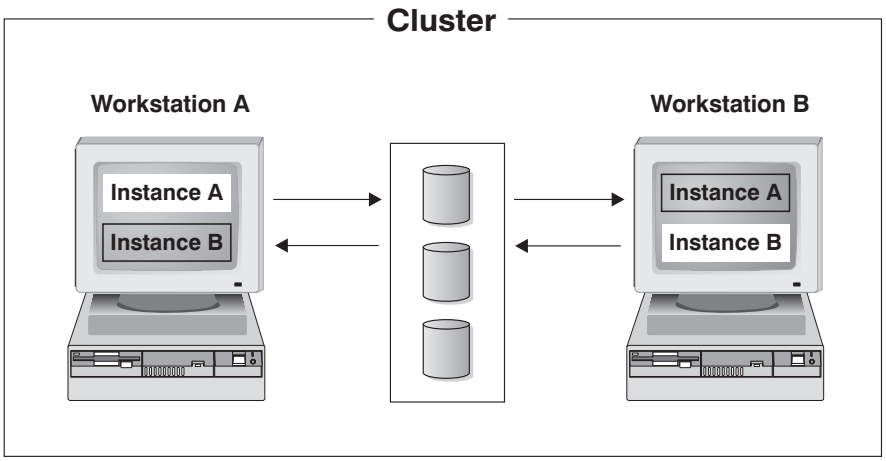


Figure 66. Mutual Takeover Configuration

Using the DB2MSCS Utility

You use the DB2MSCS utility to create the infrastructure for DB2 to support failover on the Windows NT environment using MSCS support. You can use this utility to enable failover in both single-partition and partitioned database environments.

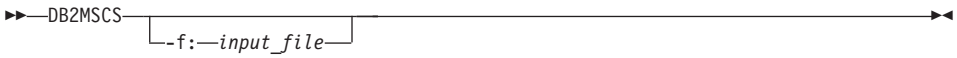
You run the DB2MSCS utility once for each instance on its instance-owning machine. If there is only one DB2 instance running on one machine in the MSCS cluster, this sets up a hot-standby configuration. If you have an instance running on each machine in the MSCS cluster, you would run DB2MSCS once on each instance-owing machine to set up a mutual takeover configuration.

The DB2MSCS utility performs the following steps:

1. Reads the required MSCS and DB2 parameters from an input file called DB2MSCS.CFG. See "Specifying the DB2MSCS.CFG File" on page 569 for information about the full set of input parameters.
2. Validates the parameters in the input file.
3. Registers the DB2 resource type.
4. Creates the MSCS group (or groups) to contain the MSCS and DB2 resources.
5. Creates the IP resource.
6. Creates the Network Name resource.

7. Moves MSCS disks to the group.
8. Creates the DB2 resource (or resources).
9. Adds all required dependencies for the DB2 resource.
10. Converts the non-clustered DB2 instance into a clustered instance.
11. Brings all resources online.

The syntax of the DB2MSCS utility is as follows:



Where:

-f:*input_file*
Specifies the DB2MSCS.CFG input file to be used by the MSCS utility. If this parameter is not specified, the DB2MSCS utility reads the DB2MSCS.CFG file that is in the current directory.

Specifying the DB2MSCS.CFG File

The DB2MSCS.CFG file is an ASCII text file that contains parameters that are read by the DB2MSCS utility. You specify each input parameter on separate line using the following format: PARAMETER_KEYWORD=*parameter_value*. For example:

```
CLUSTER_NAME=WOLFPACK
GROUP_NAME=DB2 Group
IP_ADDRESS=9.21.22.89
```

Two example configuration files are in the /CFG subdirectory of the /SQLLIB directory. The first, DB2MSCS.EE, is an example for single-partition database environments. The second, DB2MSCS.EEE, is an example for partitioned database environments.

The parameters for the DB2MSCS.CFG file are as follows:

DB2_INSTANCE
The name of the DB2 instance. If the instance name is *not* specified, the default instance (the value of the DB2INSTANCE environment variable) is used.

This parameter has a global scope, and you specify it only once in the DB2MSCS.CFG file.

This parameter is optional.

Example:

```
DB2_INSTANCE=DB2
```

The instance must already exist. For information about creating instances, refer to the *DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings*.

DB2_LOGON_USERNAME
The name of the logon account for the DB2 service.

This parameter has a global scope, and you specify it only once in the DB2MSCS.CFG file.

This parameter is only required for DB2 Enterprise - Extended Edition instances.

Example:

```
DB2_LOGON_USERNAME=db2user
```

DB2_LOGON_PASSWORD
The password of the logon account for the DB2 service. If the DB2_LOGON_USERNAME parameter is provided but the DB2_LOGON_PASSWORD parameter is not, the DB2MSCS utility prompts for the password. The password is not displayed when it is typed at the command line.

This parameter has a global scope, and you specify it only once in the DB2MSCS.CFG file.

This parameter is only required for DB2 Enterprise - Extended Edition instances.

Example:

```
DB2_LOGON_PASSWORD=xxxxxx
```

CLUSTER_NAME
The name of the MSCS cluster. All the resources specified following this line are created in this cluster until another CLUSTER_NAME tag is specified.

Specify this parameter once for each cluster.

This parameter is optional. If not specified, the name of the MSCS cluster on the local machine is used.

Example:

```
CLUSTER_NAME=WOLFPACK
```

GROUP_NAME
The name of the MSCS group. If this parameter is specified, a new MSCS group is created if it does not exist. If the group already exists, it is used as the target group. Any MSCS resource created following this line is created in this group until another GROUP_NAME keyword is specified.

Specify this parameter once for each group.

This parameter is required.

Example:

```
GROUP_NAME=DB2 Group
```

DB2_NODE

The node number of the database partition server (node) to be included in the current MSCS group. If multiple logical nodes exist on the same machine, each node requires a separate DB2_NODE keyword.

You specify this parameter after the GROUP_NAME parameter so that the DB2 resources are created in the correct MSCS group.

This parameter is only required for DB2 Enterprise - Extended Edition instances.

Example:

```
DB2_NODE=0
```

IP_NAME

The name of the IP Address resource. The value for IP_NAME is arbitrary, but must be unique. When this parameter is specified, an MSCS resource of type IP Address is created.

This parameter is required for remote TCP/IP connections. You must specify this parameter for the instance-owning machine in a partitioned database environment. This parameter is optional in single-partition database environments.

Example:

```
IP_NAME=IP Address for DB2
```

Note: DB2 clients should use the TCP/IP address of this IP resource to catalog the TCP/IP node entry. By using the MSCS IP address, when the database server fails over to the other machine, DB2 clients can still connect to the database server because the IP address is available on the fail-over machine.

The attributes of the IP resource are as follows:

IP_ADDRESS

The TCP/IP address of the IP resource. Specify this keyword to set the TCP/IP address for the preceding IP resource.

This parameter is required if the IP_NAME parameter is specified.

Example:

```
IP_ADDRESS=9.21.22.34
```

IP_SUBNET
The subnet mask for the preceding IP resource.

This parameter is required if the IP_NAME parameter is specified.

Example:

```
IP_SUBNET=255.255.255.0
```

IP_NETWORK
The name of the MSCS network that the preceding IP resource belongs to. If this parameter is not specified, the first MSCS network detected by the system is used.

This parameter is optional.

Example:

```
IP_NETWORK=Token Ring
```

NETNAME_NAME
The name of the Network Name resource. Specify this parameter to create the Network Name resource.

This parameter is optional for single-partition database environments. It is required for partitioned database environments.

Example:

```
NETNAME_NAME=Network name for DB2
```

The attributes of the Network Name resource are as follows:

NETNAME_VALUE
The value for the Network Name.

This parameter is required if NETNAME_NAME parameter is specified.

Example:

```
NETNAME_VALUE=DB2SRV
```

NETNAME_DEPENDENCY
The dependency list for the Network Name resource. Each Network Name resource must have a dependency on an IP Address resource. If this parameter is not specified, the Network Name resource has a dependency on the first IP resource in the group.

This parameter is optional.

Example:

```
NETNAME_DEPENDENCY=IP Address for DB2
```

DISK_NAME

The name of the physical disk resources to be moved to the current groups. Specify as many disk resources as you need.

Notes:

1. The disk resources must already exist.
2. When the DB2MSCS utility configures the DB2 instance for MSCS support, the instance directory is copied to the *first* MSCS disk in the group. To specify a different MSCS disk for the instance directory, use the INSTPROF_DISK parameter.

Example:

```
DISK_NAME=Disk E:
DISK_NAME=Disk F:
```

INSTPROF_DISK

An optional parameter to specify an MSCS disk to contain the DB2 instance directory. If this parameter is NOT specified, the DB2MSCS utility uses the *first* MSCS disk that belongs to the same group as the instance directory.

The DB2 instance directory is created on the MSCS disk under the X:\DB2PROFS directory (where X is the MSCS disk drive letter).

Example:

```
INSTPROF_DISK=Disk E:
```

Setting up Failover for a Single-Partition Database System

When you run the DB2MSCS utility against a single-partition database system, one MSCS group contains DB2 and all the dependent MSCS resources (the IP address, Network Name, and disks). For example, the contents of the DB2MSCS.CFG for a single-partition database system will look like the following:

```
#
#  DB2MSCS.CFG for a single-partition database system
#
DB2_INSTANCE=DB2
CLUSTER_NAME=MSCS
GROUP_NAME=DB2 Group
IP_NAME=...
IP_ADDRESS=...
IP_SUBNET=...
IP_NETWORK=...
NETNAME_NAME=...
NETNAME_VALUE=...
DISK_NAME=Disk E:
```

Setting up a Mutual Takeover Configuration for Two Single-Partition Database Systems

You can set up two single-partition database systems, each on a separate machine, so that if the database system on any one machine fails, it is restarted on the other MSCS node.

To setup failover support for this configuration, you need to run the DB2MSCS utility once on each instance-owning machine. You must tailor the configuration file for each database system.

Assume that the DB2 instances are called DB2A and DB2B. The DB2MSCS.CFG file for the DB2A instance would be as follows:

```
#
#  DB2MSCS.CFG for first single-partition database system
#
DB2_INSTANCE=DB2A
CLUSTER_NAME=MSCS
GROUP_NAME=DB2A Group
IP_NAME=...
IP_ADDRESS=...
IP_SUBNET=...
IP_NETWORK=...
NETNAME_NAME=...
NETNAME_VALUE=...
DISK_NAME=Disk E:
```

The DB2MSCS.CFG file for the DB2A instance would be as follows:

```
#
#  DB2MSCS.CFG for second single-partition database system
#
DB2_INSTANCE=DB2B
CLUSTER_NAME=MSCS
GROUP_NAME=DB2B Group
IP_NAME=...
IP_ADDRESS=...
IP_SUBNET=...
IP_NETWORK=...
NETNAME_NAME=...
NETNAME_VALUE=...
DISK_NAME=Disk F:
```

For a full example, see “Example - Setting up Two Single-Partition Instances for Mutual Takeover” on page 580.

Setting up Multiple MSCS Clusters for a Partitioned Database System

When you run the DB2MSCS utility against a multipartition database system, one MSCS group is created for each physical machine that participates in the

system. The DB2MSCS.CFG file must contain multiple sections, and each section must have a different value for the GROUP_NAME parameter and for all the required dependent resources for that group.

In addition, you must specify the DB2_NODE parameter for each database partition server in each MSCS group. If you have multiple logical nodes, each logical node requires a separate DB2_NODE keyword.

For example, assume that you have a multipartition database system that consists of four database partition servers on four machines, and you want to configure two MSCS clusters using mutual takeover configuration. You would set up the DB2MSCS.CFG configuration file as follows:

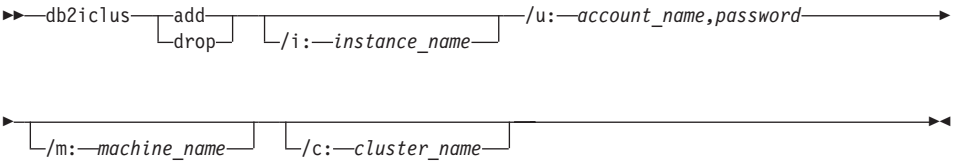
```
#
#  DB2MSCS.CFG for one partitioned database system with
#  multiple clusters
DB2_INSTANCE=DB2MPP
DB2_LOGON_USERNAME=db2user
DB2_LOGON_PASSWORD=xxxxxx
CLUSTER_NAME=MSCS1
  # Group 1
GROUP_NAME=DB2 Group 1
DB2_NODE=0
IP_NAME=...

...
# Group 2
GROUP_NAME=DB2 Group 2
DB2_NODE=1
IP_NAME=...

...

CLUSTER_NAME=MSCS2
# Group 3
GROUP_NAME=DB2 Group 3
DB2_NODE=2
IP_NAME=...

...
# Group 4
GROUP_NAME=DB2 Group 4
DB2_NODE=3
IP_NAME=...

...
```

For a full example, see “Example - Setting up a Four-Node Partitioned Database System for Mutual Takeover” on page 582.

Maintaining the MSCS System

When you run the DB2MSCS utility, it creates the infrastructure for failover support for all machines in the MSCS cluster. To remove support from a machine, use the **db2iclus** command with the **drop** option. To re-enable support for a machine, use the **add** option.

The command syntax is as follows:



Where:

add — Enables failover support on the machine by adding it to an MSCS cluster. The DB2 resource (database server) can then fail over to this machine.

drop — Removes failover support from the machine by dropping it from an MSCS cluster.

/i: *instance_name* — Is the name of the instance. (This parameter overrides the setting of the DB2INSTANCE environment variable.)

/u: *account_name*, *password* — Is the domain account used as the logon account name of the DB2 Service. For example:

```
/u:domainA\db2nt,password
```

This parameter is only required with the **add** parameter.

/m:*machine_name* — Is the computer name of the machine that you want to add to, or drop from, an MSCS cluster. You must specify this option if you run the command from a machine other than the one for which you are modifying failover support.

/c: *cluster_name* — Is the name of the MSCS cluster as it is known on the LAN. This name is specified when the MSCS cluster is first created.

Fallback Considerations

By default, groups are set not to fall back to the original (failed) machine. Unless you manually configure a DB2 group to fall back after failing over, it continues to run on the alternative MSCS node after the cause of the failover has been resolved.

If you configure a DB2 group to automatically fall back to the original machine, all the resources in the DB2 group including the DB2 resource will fall back as soon as the original machine is available. If, during the fall back, a database connection exists, the DB2 resource cannot be brought offline, and the fallback processing will fail.

If you want to force all database connections off the database during the fallback processing, set the DB2_FALLBACK registry variable to ON. This variable must be set as follows:

```
db2set DB2_FALLBACK=ON
```

You do not have to reboot or restart the cluster service after setting this registry variable.

Registering Database Drive Mapping for Mutual Takeover Configurations in a Partitioned Database Environment

When you create a database in the partitioned database environment, you can specify a drive letter for the **create database** command to indicate where the database is to be created.

Note: You do not set database drive mapping for single-partition database environments.

When the **create database** command runs, it expects that the drive that you specify will be simultaneously available to all the machines that participate in the instance. Because this is not possible, DB2 uses database drive mapping to assign the same drive a different name for each machine.

For example, assume that a DB2 instance called DB2 contains two database partition servers:

```
NODE0 is active on machine WOLF_NODE_0
NODE1 is active on machine WOLF_NODE_1
```

Also assume that the share disk E: belongs to the same group as NODE0, and that the share disk F: belongs to the same group as NODE1.

To create a database on the share disk E:, the **create database** command would be as follows:

```
db2 create database mppdb on E:
```

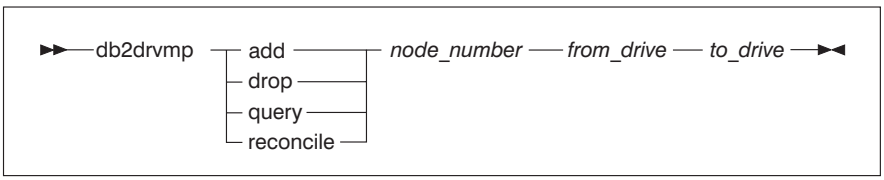
For the **create database** command to be successful, drive E: must be available to both machines. In a mutual takeover configuration, each database partition server may be active on a different machine, and the cluster disk E: is only available to one machine. In this situation, the **create database** command will always fail.

To resolve this problem, the database drive should be mapped as follows:

```
For NODE0, the mapping is from drive F: to drive E:
For NODE1, the mapping is from drive E: to drive F:
```

Any database access for NODE0 to drive F: is then mapped to drive E:, and any database access for NODE1 to drive E: is mapped to drive F:. Using drive mapping, the **create database** command will create database files on drive E: for NODE0 and drive F: for NODE1.

Use the **db2drvmp** command to set up the drive mapping. The command is as follows:



The parameters are as follows:

add Assigns a new database drive map.

drop Removes an existing database drive map.

query Queries a database map

reconcile Repairs a database map drive when the registry contents are damaged. See “Reconciling Database Drive Mapping” on page 579 for more information.

node_number The node number. This parameter is required for add and drop operations.

from_drive The drive letter to map from. This parameter is required for add and drop operations.

to_drive The drive letter to map to. This parameter is required for add operations. It is not applicable to other operations.

If you wanted to set up database drive mapping from F: to E: for NODE0, you would use the following command:

```
db2drvmp add 0 F E
```

Note: Database drive mapping does not apply table spaces, containers, or any other database storage objects.

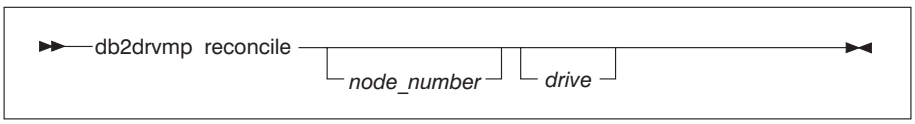
Similarly, to set up database drive mapping from drive E to drive F for NODE1, you would issue the following command:

```
db2drvmp add 1 E F
```

Note: Any setup of, or changes to, database drive mapping do not take effect immediately. To activate the database drive mapping, use the Cluster Administrator tool to bring the DB2 resource offline, then online.

Reconciling Database Drive Mapping

When a database is created on a machine that has database drive mapping in effect, the map is saved on the drive in a hidden file. This is to prevent the database drive from being removed after the database is created. To reconcile the map, run the **db2drvmp reconcile** command for each database partition server that contains the database. (A situation in which you would want to reconcile the database drive mapping would be if you accidentally dropped the database drive map.) The command syntax is as follows:



The parameters are as follows:

node_number The node number of the node to be repaired. If *node_number* is not specified, the command reconciles the mapping for all nodes.

drive The drive to reconcile. If the drive is not specified, the command reconciles the mapping for all drives.

The **db2drvmp** command scans all drives on the machine for database partitions that are managed by the database partition server, and reapplies the database drive mapping to the registry as required.

Example - Setting up Two Single-Partition Instances for Mutual Takeover

The objective for this example is to set up two single-partition database instances with failover support in a mutual takeover configuration. In this example, four servers are configured into two MSCS clusters. By using the mutual takeover configuration, when any of the machine fails, the database server configured for that machine will fail over to the alternative machine, as configured using the MSCS software, and run on the alternative machine.

There are two MSCS clusters in the resulting configuration. Each cluster has:

- Two servers, each with 64 MB of memory and one local SCSI disk of 2 GB
- One SCSI disk tower that has three shared SCSI disks of 2 GB each.

In addition, each machine has one 100X Ethernet Adapter card installed.

Each machine has the following software installed:

- Windows NT Version 4.0 Enterprise Edition with the MSCS feature installed
- DB2 Universal Database Enterprise Edition Version 6.

The resulting network configuration is as follows:

Server 1:	Server 2:
• Machine name:db2test1	• Machine name:db2test2
• TCP/IP hostname:db2test1	• TCP/IP hostname:db2test2
• IP Address: 9.9.9.1 (subnet mask: 255.255.255.0	• IP Address: 9.9.9.2 (subnet mask: 255.255.255.0
• MSCS cluster name: ClusterA	• MSCS cluster name: ClusterA

Both machines in the network are configured with TCP/IP and connected to a private LAN using an Ethernet 100 T-base Hub. In the absence of a Domain Name Server (DNS), all machines have a local TCP/IP hosts file. Each hosts file contains the following entries:

```
9.9.9.1 db2test1 # for Server 1
9.9.9.2 db2test2 # for Server 2
9.9.9.3 ClusterA # for MSCS ClusterA
9.9.9.4 db2tcp1 # for DB2 remote client connection to Server 1
9.9.9.5 db2tcp2 # for DB2 remote client connection to Server 2
```

Preliminary Tasks

Before you perform the following tasks, it is assumed that both machines belong to the same domain, called DB2NTD:

1. Create a domain account for DB2 that is a member of the local Administrators group on those machines where DB2 is going to run. Use the account for performing all tasks:
   - Set the user name to db2nt.
   - Set the password to db2nt.
2. Install the MSCS feature on the machines db2test1 and db2test2:
   - Name the MSCS cluster ClusterA.
   - The cluster IP Address is 9.9.9.3.
   - Share disk D: will be used by the MSCS software.
   - Share disks E: and F: will be used by DB2.
3. Install DB2 Universal Database Enterprise Edition Version 6 on the machine db2test1. Install the software on C:\SQLLIB, which is a local drive.
4. Install DB2 Universal Database Enterprise Edition Version 6 on the machine db2test2. Install the software on C:\SQLLIB, which is a local drive.

The next step is to set up the DB2MSCS.CFG file for each instance, then run the DB2MSCS utility for each instance.

Run the DB2MSCS Utility

To set up the db2test1 machine, perform the following tasks:

1. On the machine db2test1, log on as user db2nt. The password is db2nt.
2. Create the DB2 instance DB2A, if it does not already exist. The command to create the instance is:

   ```
   db2icrt DB2A
   ```

3. Set up the DB2MSCS.CFG file for the DB2 instance on the machine db2test1:

   ```
   #
   #  DB2MSCS.CFG for database system
   #  on machine db2test1
   DB2_INSTANCE=DB2A
   CLUSTER_NAME=ClusterA
   #
   # Group 1
   GROUP_NAME=DB2A Group
   IP_NAME=IP Address for DB2A
   IP_ADDRESS=9.9.9.4
   IP_SUBNET=255.255.255.0
   IP_NETWORK=ClusterA
   NETNAME_NAME=Network name for DB2A
   NETNAME_VALUE=DB2SRV1
   NETNAME_DEPENDENCY=IP Address for DB2A
   DISK_NAME=Disk E:
   INSTPROF_DISK=Disk E:
   ```

4. Run the DB2MSCS utility as follows:

   ```
   db2mscs -f:DB2MSCS.CFG
   ```

5. Log out from the db2nt account.
6. On the machine db2test2, log on as user db2nt, which belongs to the local Administrators group. The password is db2nt.
7. Create the DB2 instance DB2B, if it does not already exist. The command to create the instance is:

   ```
   db2icrt DB2B
   ```

8. Set up the DB2MSCS.CFG file for the DB2 instance on the machine db2test2:

   ```
   #
   #  DB2MSCS.CFG for database system
   #  on machine db2test2
   DB2_INSTANCE=DB2B
   CLUSTER_NAME=ClusterA
   #
   # Group 1
   GROUP_NAME=DB2B Group
   IP_NAME=IP Address for DB2B
   IP_ADDRESS=9.9.9.5
   IP_SUBNET=255.255.255.0
   IP_NETWORK=ClusterA
   NETNAME_NAME=Network name for DB2B
   NETNAME_VALUE=DB2SRV2
   NETNAME_DEPENDENCY=IP Address for DB2B
   DISK_NAME=Disk F:
   INSTPROF_DISK=Disk F:
   ```

9. Run the DB2MSCS utility as follows:

   ```
   db2mscs -f:DB2MSCS.CFG
   ```

10. Log out from the db2nt account.

Example - Setting up a Four-Node Partitioned Database System for Mutual Takeover

The objective for this example is to set up a four-node partitioned database system with failover support in a mutual takeover configuration. In this example, four servers are configured into two MSCS clusters. By using the mutual takeover configuration, when any of the machine fails, the database partition servers configured for that machine will fail over to the alternative machine, as configured using the MSCS software, and run as a logical node on the alternative machine.

There are two MSCS clusters in the resulting configuration. Each cluster has:

- Two servers, each with 64 MB of memory and one local SCSI disk of 2 GB
- One SCSI disk tower that has three shared SCSI disks of 2 GB each.

In addition, each machine has one 100X Ethernet Adapter card installed.

Each machine has the following software installed:

- Windows NT Version 4.0 Enterprise Edition with the MSCS feature installed
- DB2 Universal Database Extended Enterprise Edition Version 6.

The resulting network configuration is as follows:

Server 1: • Machine name:db2test1 • TCP/IP hostname:db2test1 • IP Address: 9.9.9.1 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterA	Server 2: • Machine name:db2test2 • TCP/IP hostname:db2test2 • IP Address: 9.9.9.2 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterA
Server 3: • Machine name:db2test3 • TCP/IP hostname:db2test3 • IP Address: 9.9.9.3 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterB	Server 4: • Machine name:db2test4 • TCP/IP hostname:db2test4 • IP Address: 9.9.9.4 (subnet mask: 255.255.255.0 • MSCS cluster name: ClusterB

All machines in the network are configured with TCP/IP and connected to a private LAN using an Ethernet 100 T-base Hub. In the absence of a Domain Name Server (DNS), all machines have a local TCP/IP hosts file. Each hosts file contains the following entries:

```
9.9.9.1 db2test1 # for Server 1
9.9.9.2 db2test2 # for Server 2
9.9.9.3 db2test3 # for Server 3
9.9.9.4 db2test4 # for Server 4
9.9.9.5 ClusterA # for MSCS Cluster 1
9.9.9.6 ClusterB # for MSCS Cluster 2
9.9.9.7 db2tcp # for DB2 remote client connection
```

Preliminary Tasks

Before you perform the following tasks, it is assumed that all four machines belong to the same domain, called DB2NTD:

1. Create a domain account for DB2 that is a member of the local Administrators group on those machines where DB2 is going to run. Use the account for performing all tasks:
   - Set the user name to db2nt.
   - Set the password to db2nt.
2. Create a second domain account with the *"password never expires"* characteristic. This account will be associated with DB2 services:

- Set the user name to db2mpp.
- Set the password to db2mpp.

3. Install the MSCS feature on the machines db2test1 and db2test2:
   - Name the MSCS cluster ClusterA.
   - The cluster IP Address is 9.9.9.5.
   - Share disk D: will be used by the MSCS software.
   - Share disks E: and F: will be used by DB2.
4. Install the MSCS feature on the machines db2test3 and db2test4:
   - Name the MSCS cluster ClusterB.
   - The cluster IP Address is 9.9.9.6.
   - Select share disk D: will be used by the MSCS software
   - Share disks E: and F: will be used by DB2.
5. Install DB2 Enterprise - Extended Edition on the machine db2test1:
   - Select the *"This machine will be the instance-owing database partition server"* option.
   - The account for the DB2 service is db2mpp. The password is db2mpp.
   - Install the software on C:\SQLLIB, which is a local drive.
6. Install DB2 Enterprise - Extended Edition on the machines db2test2, db2test3, and db2test4:
   - Select the *"This machine will be a new node on an existing partitioned database system"* option.
   - Select db2test1 as the instance-owning machine.
   - The account for the DB2 service is db2mpp. The password is db2mpp.
   - Install the software on C:\SQLLIB, which is a local drive.

The next step is to set up the DB2MSCS.CFG file and run the DB2MSCS utility.

Run the DB2MSCS Utility

To set up the db2test1 machine, perform the following tasks:

1. Log on as user db2nt, which belongs to the local Administrators group. The password is db2nt.
2. Set up the DB2MSCS.CFG file:

```
#
#  DB2MSCS.CFG for one partitioned database system with
#  multiple MSCS clusters
DB2_INSTANCE=DB2MPP
CLUSTER_NAME=ClusterA
DB2_LOGON_USERNAME=db2mpp
DB2_LOGON_PASSWORD=db2mpp
# Group 1
```

```
# for DB2 node 0
GROUP_NAME=DB2NODE0
DB2_NODE=0
IP_NAME=IP Address for DB2
IP_ADDRESS=9.9.9.7
IP_SUBNET=255.255.255.0
IP_NETWORK=Ethernet
NETNAME_NAME=Network name for DB2
NETNAME_VALUE=DB2WOLF
NETNAME_DEPENDENCY=IP Address for DB2
DISK_NAME=Disk E:
INSTPROF_DISK=Disk E:
#
# Group 2
# for DB2 node 1
GROUP_NAME=DB2NODE1
DB2_NODE=1
DISK_NAME=Disk F:
#
CLUSTER_NAME=ClusterB
# Group 3
# for DB2 node 2
GROUP_NAME=DB2NODE2
DB2_NODE=2
DISK_NAME=Disk E:
#
# Group 4
# for DB2 node 3
GROUP_NAME=DB2NODE3
DB2_NODE=3
DISK_NAME=Disk F:
```

3. Run the DB2MSCS utility as follows:

   ```
   db2mscs -f:DB2MSCS.CFG
   ```

4. Log out from the db2nt account.

The final steps are to register the database drive mapping for the two MSCS clusters.

Register the Database Drive Mapping for ClusterA

To register the database drive mapping for MSCS cluster ClusterA, perform the following tasks:

1. On the machine db2test1, log on as user db2mpp, which is the account associated with DB2 services. The password is db2mpp.
2. To register the database drive mapping, enter the following commands:

   ```
   db2drvmp add 0 F E

   db2drvmp add 1 E F
   ```

3. Bring all DB2 resources offline, then bring them online.

Register the Database Drive Mapping for ClusterB

To register the database drive mapping for MSCS cluster `ClusterB`, perform the following tasks:

1. On the machine `db2test3`, log on as user `db2mpp`, which is the account associated with DB2 services. The password is `db2mpp`.
2. To register the database drive mapping, enter the following commands:

   ```
   db2drvmp add 2 F E

   db2drvmp add 3 E F
   ```

3. Bring all DB2 resources offline, then bring them online.

Administering DB2 in an MSCS Environment

If you are using MSCS clusters, your DB2 instance requires additional planning with regards to daily operation, database deployment, and database configuration. For DB2 to execute transparently on any MSCS node, additional administrative tasks must be performed. All DB2 dependent operating system resources must be available on all MSCS nodes. Some of these operating system resources fall outside the scope of MSCS. That is, they cannot be defined as an MSCS resource. You must ensure that each system is configured such that the same operating system resources are available on all MSCS nodes. The sections that follow describe the additional work that must be done.

Starting and Stopping DB2 Resources

You must start and stop DB2 resources from the Cluster Administrator tool. Several mechanisms are available to start a DB2 instance such as the **db2start** command, and the **Services** option from the Control Panel. However, if DB2 is not started from the Cluster Administrator, the MSCS software will not be aware of the state of DB2 instance. If a DB2 instance is started using the Cluster Administrator and stopped using the **db2stop** command, the MSCS software will interpret the **db2stop** command as a software failure and attempt to restart DB2. (The current MSCS interfaces do not support notification of a *resource state*.)

Similarly, if you use **db2start** to start a DB2 instance, MSCS cannot detect that the resource is online. If a database server failed, MSCS would not bring the DB2 resource online on the failover machine in the cluster.

Three operations can be applied to a DB2 instance:

Online
This operation is equivalent to using the **db2start** command. If DB2 is

already active, this operation can be used simply to notify MSCS that DB2 is active. Any errors during this operation will be written to the Windows NT Event Log.

Offline
This operation is equivalent to using the **db2stop** command. If there are any active connections to an instance, this operation will fail. This is consistent with the behavior of **db2stop**.

Fail resource
This operation is equivalent to using the **db2stop** command with the **force** option specified. DB2 will disconnect all applications off the DB2 system and stop all database servers.

Running Scripts

You can execute scripts both before and after a DB2 resource is brought online. These scripts *must* reside in the instance profile directory that is specified for the DB2INSTPROF environment variable. This directory is the directory path that is specified by the **-p** parameter of the **db2icrt** command. You can obtain this value by issuing the following command:

```
db2set -i:instance_name DB2INSTPROF
```

This file path must be on a clustered disk so that the instance directory is available on all cluster nodes.

These script files are not required, and are only executed if they are found in the instance directory. They are launched by the MSCS Cluster Service in the background. The script files must redirect standard output to record any output as a result of commands within the script file. The output is not displayed to the screen.

In a partitioned database environment, by default, the same script will be used by every database partition server in the instance. If you need to distinguish among the different database partition servers in the instance, use different assignments of the DB2NODE environment variable to target specific node numbers (for example, use the IF statement in the db2cpre.bat and db2cpost.bat files).

Running Scripts Before Bringing DB2 Resources Online

If you want to run a script before you bring a DB2 resource online, the script *must* be named db2cpre.bat. DB2 calls functions that will launch this batch file from the Windows NT command line processor and wait for the command line processor to complete execution before the DB2 resource is brought online. You can use this batch file for tasks such as modifying the DB2 database manager configuration. You may want to change some database

manager parameter values if the failover system is constrained, and you must reduce the system resources consumed by DB2.

The commands placed in the db2cpre.bat script should execute synchronously. Otherwise the DB2 resource may be brought online before all tasks in the script are completed, which may result in unexpected behavior. Specifically, **db2cmd** should not be executed in the db2cpre.bat script, because it, in turn, launches another command processor, which will execute DB2 commands asynchronously to the **db2cmd** program.

If you want to use DB2 CLP commands in the db2cpre.bat script, the commands should be placed in a file and executed as a CLP batch file from within a program that initializes the DB2 environment for the DB2 command line processor, then waits for the completion of the DB2 command line processor. For example:

```
#include <windows.h>

int WINAPI DB2SetCLPEnv_api(DWORD pid);

void main (int argc, char *argv [ ] )
{
      STARTUPINFO         startInfo   = {0};
      PROCESS_INFORMATION pidInfo     = {0};
      char     title  [32]   = "Run Synchronously";
      char     runCmd [64]  =
                             "DB2 -z c:\\run.out -tvf c:\\run.clp";

/* Invoke API to setup a CLP Environment */
      if ( DB2SetCLPEnv_api (GetCurrentProcessId ()) == 0 )(1 - see notes below)
      {
         startInfo.cb          = sizeof(STARTUPINFO);
         startInfo.lpReserved  = NULL;
         startInfo.lpTitle     = title;
         startInfo.lpDesktop   = NULL;
         startInfo.dwX         = 0;
         startInfo.dwY         = 0;
         startInfo.dwXSize     = 0;
         startInfo.dwYSize     = 0;
         startInfo.dwFlags     = 0L;
         startInfo.wShowWindow = SW_HIDE;
         startInfo.lpReserved2 = NULL;
         startInfo.cbReserved2 = 0;
               if ( CreateProcessA( NULL,
                              runCmd, (2)
                              NULL,
                              NULL,
                              FALSE,
                              NORMAL_PRIORITY_CLASS ³ CREATE_NEW_CONSOLE,
                              NULL,
                              NULL,
                              &startInfo,
                              &pidInfo))
```

```
         {
            WaitForSingleObject (pidInfo.hProcess, INFINITE);
            CloseHandle (pidInfo.hProcess);
            CloseHandle (pidInfo.hThread);
         }
      }
      return;
}
```

Notes:

1. The API DB2SetCLPEnv_api is resolved by the import library DB2API.LIB. This API sets an environment that allows CLP commands to be invoked. If this program is invoked from the db2cpre.bat script, the command processor will wait for the CLP commands to complete.
2. runCmd is the name of the script file that contains the DB2 CLP commands.

A sample program called db2clpex.exe can be found in the MISC subdirectory of the DB2 install path. This executable is similar to the example provided, but accepts the DB2 CLP command as a command line argument. If you want to use this sample program, copy it to the BIN subdirectory. You can use this executable in the db2cpre.bat script as follows (INSTHOME is the instance directory).

```
db2clpex "DB2 -Z INSTHOME\pre.log -tvf INSTHOME\pre.clp"
```

All DB2 **attach** commands or **connect** statements should explicitly specify a user, otherwise they will be executed under the user account associated with the cluster service. CLP scripts should also complete with the **terminate** command to end the CLP background process.

The following is an example of a db2cpre.bat file:

```
db2cpre.bat : (1 - see notes below)
-----------------------
db2clpex "db2 -z INSTHOME\pre-%DB2NODE%.log (2, 3)
   -tvf INSTHOME\pre.clp" (4, 5)
-----------------------

PRE.CLP (6)
-----------------------
update dbm cfg using MAXAGENTS 200;
get dbm cfg;
terminate;
-----------------------
```

Notes:

1. The db2cpre.bat script executes under the user account associated with the Cluster Service. If DB2 actions are required, the user account associated with the Cluster Service must be a valid SQL identifier, as defined by DB2.
2. INSTHOME is the instance directory.

3. The name of the log file must be different for each node to avoid file contention when both logical nodes are brought online at the same time.
4. db2clpex.exe is the sample program previously provided using the command line argument to specify the CLP command to execute. (This line of the example has been split at `-tvf` for formatting reasons.)
5. The db2clpex.exe sample program must be made available on all MSCS cluster nodes.
6. The CLP commands in this example set a limit on the number of agents.

Running Scripts After Bringing DB2 Resources Online

If you want to run a script after you bring a DB2 resource online, it *must* be named db2cpost.bat. The script will be executed asynchronously from MSCS after the DB2 resource has been successfully brought online. The **db2cmd** command can be used in this script to execute DB2 CLP script files. Use the **-c** parameter of the **db2cmd** command to specify that the utility should close all windows on completion of the task. For example:

```
db2cmd -c db2 -tvf mycmds.clp
```

The **-c** parameter must be the first argument to the **db2cmd** command, as it prevents orphaned command processors in the background.

The db2cpost.bat script is useful if you want to perform database activities immediately after the DB2 resource fails over and becomes active. For example, you can restart or activate databases in the instance so that they are primed for user access.

The following is an example of a db2cpost.bat script:

```
db2cpost.bat (1 - see notes below)
-----------------------
db2cmd -c db2 -z INSTHOME\post-%DB2NODE%.log (2, 3)
   -tvf INSTHOME\post.clp (4)
-----------------------

POST.CLP (5)
-----------------------
restart database SAMPLE;
connect reset;
activate database SAMPLE;
terminate;
-----------------------
```

Notes:

1. The db2cpost.bat script executes under the user account associated with the Cluster Service. If DB2 actions are required, the user account associated with the Cluster Service must be a valid SQL identifier, as defined by DB2.

2. INSTHOME is the instance directory.
3. The name of the log file must be different for each node to avoid file contention when both logical nodes are brought online at the same time.
4. The **db2cmd** command can be used because the db2cpost.bat script can execute asynchronously. The **-c** parameter must be used to terminate the command processor.
5. The CLP script in this example contains commands to restart and activate the database. This script returns the database to an active state immediately after the database manager is started. In a partitioned database system you should remove the **activate database** command because multiple DB2 resources are brought online at the same time: the **restart database** command may fail because another node is activating the database. If this occurs, rerun the script to ensure that the database is restarted correctly.

Database Considerations

When you create a database, ensure that the database path refers to a share disk. This allows the database to be seen on all MSCS nodes. All logs and other database files must also refer to clustered disks for DB2 to failover successfully. If you do not perform these steps, a DB2 system failure will occur as it will seem to DB2 that files have been deleted or are unavailable.

Also ensure that the database manager and database configuration parameters are set so that amount of system resources consumed by DB2 is supported on either MSCS node. The *autorestart* database configuration parameter should be set to ON so that the first database connection on failover will bring the database to a consistent state. The default setting for *autorestart* is ON. The database can also be brought to a ready state by using the db2cpost.bat script to restart and activate the database. This method is preferred, because there will be no dependency on *autorestart*, and the database is brought to a ready state independent of a user connection request.

User and Group Support

DB2 relies on Windows NT for user authentication and group support. For a DB2 instance to fail over from one MSCS node to another in a seamless fashion, each MSCS node must have access to the same Windows NT security databases. You can achieve this by using Windows NT Domain Security.

Define all DB2 users and groups in a Domain Security database. The MSCS nodes must be members of this Domain or the Domain must be a Trusted Domain. DB2 will then use the Domain Security database for authentication and group support, independent of which MSCS node DB2 is executing on.

If you are using local accounts, the accounts must be replicated on each MSCS node. This approach is not recommended because it is error prone and requires dual maintenance.

DCE Security is also a supported authentication mode, providing that all MSCS nodes are clients in the same DCE cell.

You should associate the MSCS service with a user account that follows DB2 naming conventions. This allows the MSCS service to perform actions against DB2 that may be required in the db2cpre.bat and db2cpost.bat scripts.

Communications Considerations

DB2 supports two LAN protocols in an MSCS Environment:

- TCP/IP
- NetBIOS

TCP/IP is supported because it is a supported cluster resource type. To enable DB2 to use TCP/IP as a communications protocol for a partitioned database system, create an IP Address resource and place it in the same group as the DB2 resource that represents the database partition server that you want to use as a coordinator node for remote applications. Then create a dependency using the Cluster Administrator tool to ensure that the IP resource is online before the DB2 resource is started. DB2 clients can then catalog TCP/IP node directory entries to use this TCP/IP address.

The TCP/IP port associated with the *svcename* database manager configuration parameter must be reserved for use by the DB2 instance on all machines that participate in the instance. The service name associated with the port number must also be the same in the services file on all machines.

Although NetBIOS is not a supported cluster resource, you can use NetBIOS as a LAN protocol because the protocol ensures that NetBIOS names are unique on the LAN. When DB2 registers a NetBIOS name, NetBIOS ensures the name is not in use on the LAN. In a failover scenario, when DB2 is moved from one system to another, the *nname* used by DB2 will be deregistered from one partner machine in the MSCS cluster and registered on the other machine.

DB2 NetBIOS support uses NetBIOS Frames (NBF). This protocol stack can be associated with different logical adapter numbers (LANA). To ensure consistent NetBIOS access to the server, the LANA associated with the NBF protocol stack should be the same on all clustered nodes. You can configure this by using the **Networks** option from the Control Panel. You should associate NBF with LANA 0, as this is the default setting expected by DB2.

System Time Considerations

DB2 uses the system time to timestamp certain operations. All MSCS nodes that participate in DB2 failover must have the system time zone and system time synchronized to ensure DB2 behaves consistently on all machines.

Set the system time zone using the **Date/Time** option from the Control Panel dialog. MSCS has a time service that synchronizes the date and time when the MSCS nodes join to form a cluster. The time service, however, only synchronizes the time every 12 hours, which may result in problems if the time is changed on one system and DB2 fails over before the time is synchronized.

If the date/time is changed on one of the MSCS cluster nodes, the time should be manually synchronized on the other cluster nodes using the command:

```
net time /set /y \\remote node
```

Where *remote node* is the machine name of the cluster node.

Administration Server and Control Center Considerations in a Partitioned Database Environment

The DB2 Administration Server is (optionally) created during the installation of DB2 Universal Database. It is not a partitioned database system. The Control Center uses the services provided by the Administration Server to administer DB2 instances and databases.

In a partitioned database system, a DB2 instance can reside on multiple MSCS nodes. This implies that a DB2 instance must be cataloged on multiple systems under the Control Center so that the instance remains accessible, regardless of which MSCS node the DB2 instance is active on.

The Administration Server instance directory is not shared. You must mirror all user-defined files in the Administration Server directory to all MSCS nodes to provide the same level of administration to all MSCS nodes. Specifically, you must make user scripts and scheduled executables available on all nodes. You must also ensure that scheduled activities are scheduled on all machines in an MSCS cluster.

Alternatively, instead of duplicating the Administration Server on all machines, you may want to have the Administration Server fail over. For the purposes of the following example, assume that you have two MSCS nodes in the cluster, and they are called MACH0 and MACH1. MACH0 has access to a cluster disk that will be used by the Administration Server. Also assume that

both MACH0 and MACH1 have an Administration Server. You would perform the following steps to make the Administration Server highly available:

1. Stop the Administration Server on both machines by executing the **db2admin stop** command on each machine.
2. On all administration client machines, uncatalog all references to the Administration Servers on MACH0 and MACH1 using the **db2 uncatalog node** command. (You can use the **db2 list node directory** command on the client machine to determine if any references to the Administration Server exist.)
3. Drop the Administration Server from MACH1 by executing the **db2admin drop** command from MACH1. (You would only perform this step if you had an Administration Server on both machines.)
4. Determine the name of the Administration Server by issuing the **db2admin** command from MACH0. (The default name is DB2DAS00.)
5. Use the DB2MSCS utility to set up fail-over support for the Administration Server. This entails creating a DB2 resource on MSCS named DB2DAS00 that has dependencies on the IP and disk resources. (If you have a mutual takeover configuration, you would put the resource in the group that holds the DB2 resource for NODE0.) This resource will be used as the MSCS resource that supports the Administration Server. The DB2MSCS.ADMIN file would be as follows:

   ```
   #
   # db2mscs.admin for Administration Server
   # run db2mscs -f:db2mscs.admin
   #
   DB2_INSTANCE=DB2DAS00
   CLUSTER_NAME=CLUSTERA
   DB2_LOGON_USERNAME=db2admin
   DB2_LOGON_PASSWORD=db2admin
   # put Administration server in the same group as DB2 Node 0
   GROUP_NAME=DB2NODE0 (see note below)
   DISK_NAME=DISK E:
   INSTPROF_DISK=DISK E:
   IP_NAME= IP Address for Administration Server
   IP_ADDRESS=9.9.9.8
   IP_SUBNET=255.255.255.0
   IP_NETWORK=Ethernet
   ```

   Note: This group can be the same as the existing group. This way, you do not require an additional disk for the instance profile directory.
6. On MACH1 execute the following command to set DB2DAS00 as the Administration Server:

   ```
   db2set -g db2adminserver=DB2DAS00
   ```

7. On MACH0, modify the start-up properties of DB2DAS00 through the Services program so that it is brought up manually and not automatically, because DB2DAS00 is now controlled by MSCS.

When the Administration Server is enabled for failover, all remote access should use an MSCS IP resource for communicating with the Administration Server. The Administration Server will now have the following properties:

- The Administration Server instance directory will fail over with the Administration Server.
- Clients will only catalog a single node to communicate with the Administration Server, regardless of which MSCS node it is active on.
- Jobs only need to be scheduled once against the Administration Server.
- Local instances can only be controlled by the Administration Server when the Administration Server is active on the same MSCS node as the local instance.
- The Administration Server is not accessible if the Cluster Service is not active.

Limitations and Restrictions

When you run DB2 in an MSCS environment:

- You cannot use physical I/O on shared disks, unless the shared disks have the same physical disk number across both MSCS nodes. You can use logical I/O because the disk is accessed using a partition identifier.
- You must configure all DB2 resource for MSCS support. If you do not, system errors will occur during DB2 runtime (DB2 cannot properly operate in the absence of system resources). For example, if the database logs are not on a MSCS shared disk, DB2 cannot restart the database.
- You must manage a DB2 instance from the Cluster Administrator tool. MSCS will view other mechanisms that are used to start and stop the database manager as software inconsistencies. For example, if you use MSCS to start DB2 and the **db2stop** command to stop DB2, MSCS will detect this as a software failure and will restart the instance. This also means that you should not use the Control Center to start and stop DB2.
- To uninstall DB2, you must first stop MSCS.

Chapter 15. High Availability in the Solaris Operating Environment, Single-Partition Database

You can set up your database system so that if a machine fails, the database server on it can run on another machine. On the Solaris Operating Environment, you implement failover support with Sun Cluster 2.1.

Sun Cluster 2.1 performs both failure detection and the restarting of resources in a clustered environment, as well as failover support for physical disks and IP addresses.

You can implement failover support for both single-partition and partitioned database systems. For details on setting up failover support for a single-partition database system, see "Setting up Failover Support for a Database System" on page 601. For details about setting up failover support for a partitioned database system, see "Chapter 16. High Availability in the Solaris Operating Environment, Partitioned Database" on page 607. For information about clients and clustered environments, see "Client Application Considerations" on page 606.

Note: Do not use a **kill -9** against the **db2start** process in a high availability environment. This action is not recommended in any environment, but in particular such an action may invalidate failover recovery in your high availability environment.

Cluster Components

Each cluster consists of the following components and resources:

Physical machine
Each physical machine has one public network interface, one or more private network interfaces on the public network, a set of shared disks, and a disk for the operating system. Each cluster can contain up to four physical machines.

Logical host
The logical host essentially *borrows* the CPU or (CPUs) and memory from the physical machine, and migrates from machine to machine during a failover situation. Each logical host consists of the following resources:

- One logical interface on the public network with its own IP *address* and *hostname*

Remote clients should always use this IP address when connecting to failover services, because the address is moved from one machine to another machine during a failover.

- One or more disk groups

  The disk group (or disk set) is a collection of physical disks that are associated with the logical host. The disk group must be on disks that are physically shared between the two machines in a cluster.

- One high availability service (that is, DB2).

  The high availability service provides a set of scripts that Sun Cluster 2.1 can use to start, stop, and abort the service.

  When a failover occurs, the high availability service from one machine fails over to another machine. You must ensure that the physical machine has enough CPU and memory resource to properly run the system after the failover; otherwise, the services may fail.

You can have as many logical hosts as you want on a machine, but for administrative reasons, it is recommended that you assign no more than one to a machine. The following is an example of the layout for a logical host filesystem for Sun Cluster 2.1 with DB2. The name of the logical host in this example is `snap`, and the DB2 instance is `DB2INST`:

/snap/ The logical host filesystem (needed for Sun Cluster 2.1).

/snap/home/DB2INST
The place to put the high availability instance home directory.

/snap/disks/DB2INST
The place to put SMS filesystems.

You only need to set up the directories */*_logical_host_name_/home/_DB2_instance_ and */*_logical_host_name_/disks/_DB2_instance_ on one logical host in the cluster.

Private network
Private networks are used for communicating between two nodes. Heartbeat messages as well as Remote Procedure Calls (RPCs) travel over these networks to keep the two nodes in synchronous operation so that they can back up each other in the event of a failover.

Public networks
The public network includes all the primary and logical network interfaces and IP addresses. The logical network interfaces or logical hosts should be referred to when communicating with DB2 on the cluster.

Disk group
Disk groups contain one or more shared disks and a list of hosts which can access these disks. Only one host can own the disk sets for exclusive use at a time.

Disk mirroring
It is highly recommended that you mirror disks to increase disk availability.

Figure 67 shows an example of the components in a cluster.

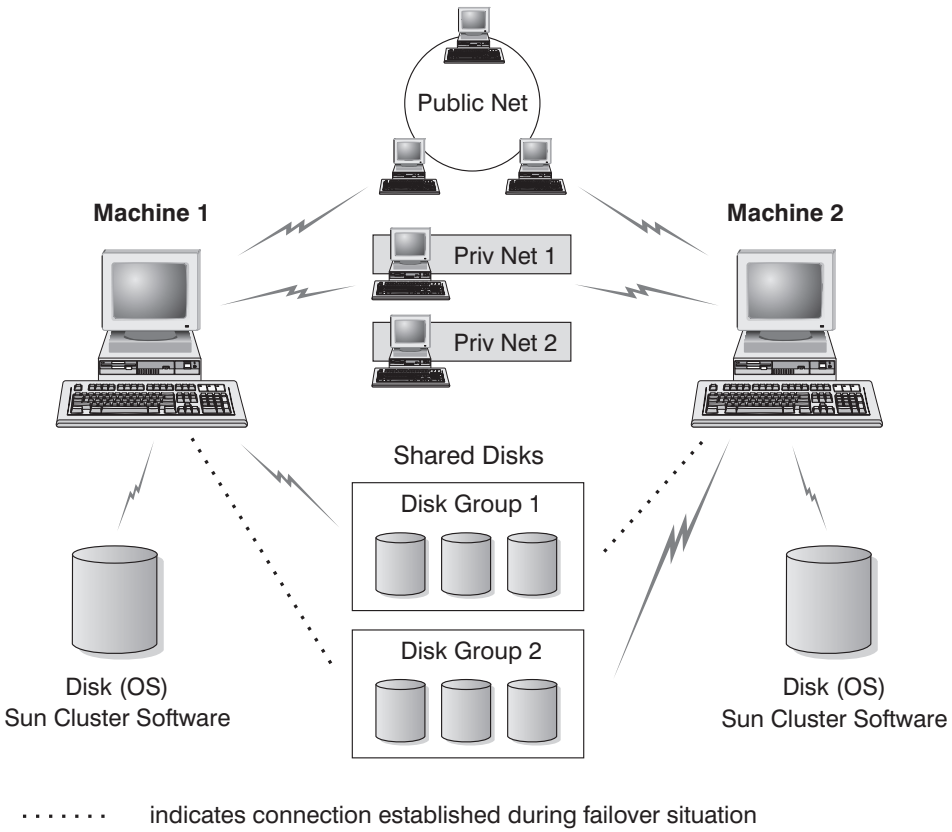


Figure 67. Components in a Cluster

The following sections describe the different types of failover support, and how to implement them.

Failover Configurations

Two types of configuration are available in a DB2 system:

- Hot standby (asymmetric mode)
- Mutual takeover (symmetric mode)

Hot Standby Configuration

In a hot standby configuration, one machine in the cluster provides dedicated backup support, and a database server runs on the other machine. If the machine participating in the database system fails, the database server on it will be started on the backup machine.

If, in a partitioned database system, you are running multiple logical nodes on a machine and it fails, they will be started on the backup machine.

Figure 68 shows an example of a hot standby configuration.

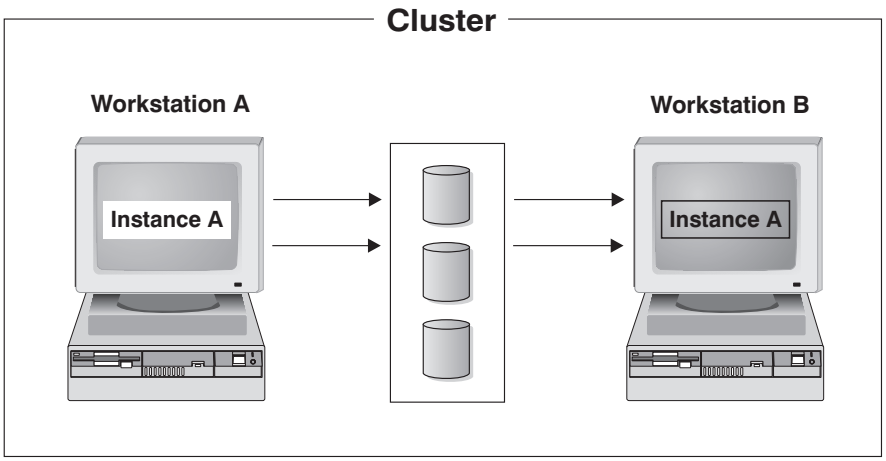


Figure 68. Hot Standby Configuration

If you are using a hot standby configuration, you can use the failover machine to run applications other than DB2.

Mutual Takeover Configuration

In a mutual takeover configuration, both machines may be running a database system (that is, each machine has at least one database server running on it). You would also see this situation with a partitioned database system, in which a database partition server would be running on each machine in the cluster. If one of the machines in the cluster fails, the database server on the failing machine is started to run on the other machine. In a mutual takeover

configuration, a database server on one machine can fail independently of the database server on another machine. Any database server can be active on any machine at any given point in time. Figure 69 shows an example of a mutual takeover configuration.

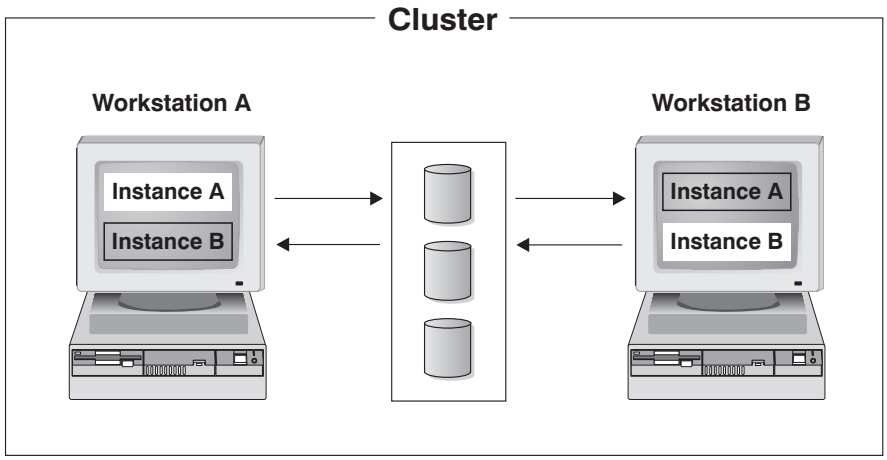


Figure 69. Mutual Takeover Configuration

Setting up Failover Support for a Database System

To set up Sun Cluster 2.1, perform the following steps:

1. "Choosing a Failover Configuration".
2. "Creating a DB2 Instance" on page 602.
3. "Registering the DB2 Resource with Sun Cluster" on page 604.
4. "Enable Failover for an Instance" on page 605.
5. "Starting and Stopping DB2" on page 605.

If you want to remove failover support for DB2, see "Unregistering DB2 for Failover" on page 606.

Choosing a Failover Configuration

To choose a failover configuration, perform the following steps:

1. Set up the machines to use either a hot standby or mutual takeover configuration. For a hot standby configuration, use one logical host. For a mutual takeover, use two logical hosts.
2. Decide on the amount of disk space that is required for each logical host and its resources, such as raw devices or SMS table space containers.

Whether you use SMS or DMS (raw devices) table spaces, any disks belonging to a logical host must be included in its disk sets.

Table space considerations: You must decide on the type of table space that you want to use. If you want to use SMS table spaces, you must set them up using disks from the disk groups that belong to a logical host. In addition, you must include them in the vfstab for the logical host. Refer to the Sun Cluster 2.1 documentation for information about how to add a file system to a logical host.

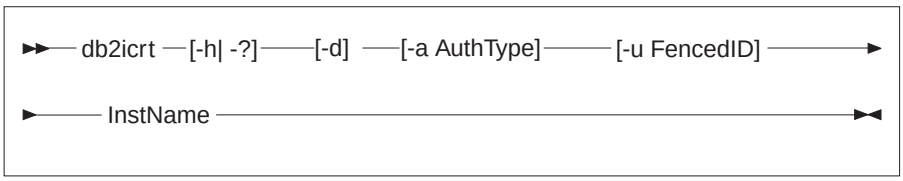
There are benefits and costs associated with using either SMS or DMS table spaces. For example, SMS table spaces reside on file systems that must be file-system checked before they are mounted. This can add a considerable amount of overhead when failover occurs, and can result in the Sun Cluster 2.1 software timing out. If you use SMS table spaces, ensure that they are journaled files systems, which require less time to check after a failover.

DMS table spaces do not have to be file-system checked during failover. This can reduce the failover time for the high availability scripts, but you should remember that committed transactions that are written to the logs will be applied to the database during crash recovery after the database server fails over.

Creating a DB2 Instance

To create the instance, use the **db2icrt** command, which is located in the DB2DIR/instance directory, where DB2DIR is /opt/IBMdb2/V5.0. Before creating a DB2 instance, ensure that DB2 is installed on each machine in the cluster.

You only create a DB2 instance on the logical host in the cluster where you created the subdirectories */logical_host_name*/home and */logical_host_name*/disks. The syntax of the **db2icrt** command is:



where:

-h or -? Display a help menu for this command.

-d Sets the debug mode that you can use for problem determination.

-a *AuthType* Is an optional parameter that specifies the authentication type for the instance. Valid authentication types are SERVER, CLIENT, and DCS. If the *-a* parameter is not specified, the

authentication type defaults to SERVER, if a DB2 server is installed. Otherwise, the *AuthType*is set to CLIENT.

Notes:

1. All databases in the instance have the same authentication type.
2. DCE authentication is not valid for this command; however, you can enable DCE authentication for an instance. For more information, refer to the *Administration Guide*.

-u *FencedID* Is the user under which the fenced UDFs and stored procedures will execute. This is not required if you install the DB2 Client or the DB2 Software Developer's Kit. For other products, this is a required parameter.

Note: *FencedID* may not be *root* or *bin*.

InstName Is the login name of the instance owner.

When you create an instance, ensure that its primary and secondary groups are different from the Administration Server's primary (SYSADM) group. When you create an instance on the same machine as the Administration Server, its SYSADM group is automatically added to the secondary group list of the Administration Server so that you can use the Control Center to perform administration tasks on that instance.

To create an instance for a DB2 server, you can use the following command:

```
db2icrt -u db2fenc1 db2inst1
```

When an instance is created, its name is also added to the list of instances on the system.

The **db2icrt** command creates the *INSTHOME*/sqllib directory, where *INSTHOME* is the home directory of the instance owner.

Note: To avoid a potential loss of data if an instance is deleted, you should not create user files or directories under the INSTHOME/sqllib directory, other than those created by DB2. The exception is if your system supports fenced user defined functions and fenced stored procedures. In this situation, put the fenced applications in the INSTHOME/sqllib/function directory.

Registering the DB2 Resource with Sun Cluster

Use the **db2hareg** script as an example of how to register DB2 with Sun Cluster. The script is located in the /opt/IBMdb2/V5.0/ha/UDB-EE_SC2.x/bin directory. The **db2hareg** script is as follows:

```
#!/bin/ksh

hareg -r db2hareg -b /opt/IBMdb2/V5.0/ha/UDB-EE_SC2.x/bin -m
START_NET=hadb2ee_startnet,STOP_NET=hadb2ee_stopnet,ABORT_NET=hadb2ee_abortnet -t
START_NET=600,STOP_NET=600
-h log0
hareg -y hadb2ee
```

In the sample, `log0` is the logical host. Replace `log0` with the logical host that is to host the DB2 services.

You should run the **db2hareg** script (or an equivalent script) once for the cluster, and you must ensure that the script is the same on both machines in the cluster. You run the script as `root`. The script both registers DB2 for failover support, and enables the following scripts for the cluster:

- **/var/db2/v5/db2tabee**

  As `root`, you must create and edit this configuration file. You must also ensure that the configuration file exists (and is the same) on both machines in the cluster. You use the configuration file to enable specific instances for failover. See “Enable Failover for an Instance” on page 605 for details.

- The following scripts are run if they exist and are executable. Both scripts only take one argument, which is the number of logical hosts that are currently being hosted. See “Running Scripts During a Failover” on page 605 for details.

  – **/var/db2/v5/failover.ee**

    This script runs at the very beginning of a failover situation.

  – *$INSTHOME*/**sqllib/ha/pre_db2startee**

    This file runs immediately before the **db2start** command

  – *$INSTHOME*/**sqllib/ha/pre_db2stopee**

    This file runs immediately before the **db2stop** command

    Note: This script may not be run if the machine crashes.

  – *$INSTHOME*/**sqllib/ha/post_failoveree**

    This file runs just after a failover and is used to restart databases.

Enable Failover for an Instance

To enable an instance for failover, you create an entry for it in the /var/db2/v5/db2tabee file. The file must exist on each machine in the cluster. The file takes entries of the form:

```
TYPE INSTANCE LOGICAL_HOST ON/OFF
```

Where:

TYPE Is the type of instance. The value can be one of the following:

- DATA to indicate a database instance.
- ADMIN to indicate an administration server instance.

INSTANCE
Is the user name of the instance owner.

LOGICAL_HOST
Is the logical host on which the DB2 instance runs.

ON/OFF
Specifies whether the instance is highly available (ON) or not (OFF).

Starting and Stopping DB2

To start DB2 in a failover environment, use the **hareg -y hadb2ee** command. This command both enables the failover environment, and starts DB2.

If you want to stop DB2, first issue the **hareg -n hadb2ee** command to disable the failover environment. Then issue the **db2stop** command to stop DB2.

Note: If you do not issue **hareg -n hadb2ee** first, Sun Cluster 2.1 may assume that the DB2 instance needs to be failed over.

Running Scripts During a Failover

The **/var/db2/v5/failover.ee** script runs automatically when a failover occurs. You can use this script to send email (for example, to notify support staff) of the failover situation. You should keep the commands in this script to a minimum, because it runs before DB2 is started. Depending on whether DB2 is starting or stopping, the following scripts will also run (if they are available) for each instance:

- *$INSTHOME***/sqllib/ha/pre_db2startee**

  This file takes as an argument the number of logical hosts that are currently running on the failover machine. If this script exists, it runs immediately before the **db2start** command.
- *$INSTHOME***/sqllib/ha/pre_db2stopee**

This file takes as an argument the number of logical hosts that are currently running on the failover machine. If this script exists, it runs immediately before the **db2stop** command.

Note: This script may not be run if the machine crashes.

- *$INSTHOME*/**sqllib/ha/post_failoveree**

  This file runs just after a failover and is used to restart databases.

You can use **pre_db2startee** to prevent resource contention by adjusting database manager and database configuration parameters that may consume substantial amounts of resource (for example, *sheapthres*). The following is an example:

```
#!/bin/ksh

#Very simple example

LOGHOSTS=$1

if [[ $LOGHOSTS -eq 1 ]]
then
  db2 update dbm cfg using SHEAPTHRES 40000
else
  db2 update dbm cfg using SHEAPTHRES 20000
fi
```

Unregistering DB2 for Failover

To unregister DB2 for failover, run the **hadb2ee.unreg** script. This script deregisters DB2 with Sun Cluster 2.1.

Client Application Considerations

Client applications should communicate with the high availability services only through the logical hostname of the logical host of the high availability service. You should ensure that client applications are written to accept a communications error and possibly retry after a few minutes.

Consider a typical client connection. The client is connected to machineA through the logical host called snap. If machineA fails, then snap fails over to machineB. According to machineB, the client connection does not exist, and will send the client a connect reset message, which will appear to the client as a communication error. The client must reconnect to the server to obtain a new connection from machineB when DB2 starts.

Chapter 16. High Availability in the Solaris Operating Environment, Partitioned Database

Sun Cluster 2.1 provides increased availability through clusters of servers that share resources such as disks and network access. If one server fails then another in the cluster can substitute for the failed one.

Note: Do not use a "kill -9" against the db2start process in a high availability environment. This action is not recommended in any environment, but in particular such an action may invalidate failover recovery in your high availability environment.

Cluster Components

Each cluster consists of the following components and resources:

Physical machine
Each physical machine has one public network interface, one or more private network interfaces on the public network, a set of shared disks, and a disk for the operating system. Each cluster can contain up to four physical machines.

Logical host
The logical host essentially *borrows* the CPU or (CPUs) and memory from the physical machine, and migrates from machine to machine during a failover situation. Each logical host consists of the following resources:

- One logical interface on the public network with its own IP *address* and *hostname*

  Remote clients should always use this IP address when connecting to failover services, because the address is moved from one machine to another machine during a failover.

- One or more disk groups

  The disk group (or disk set) is a collection of physical disks that are associated with the logical host. The disk group must be on disks that are physically shared between the two machines in a cluster.

- One high availability service (that is, DB2)

  The high availability service provides a set of scripts that Sun Cluster 2.1 can use to start, stop, and abort the service.

  When a failover occurs, the high availability service from one machine fails over to another machine. You must ensure that the

physical machine has enough CPU and memory resource to properly run the system after the failover; otherwise, the services may fail.

You can have as many logical hosts as you want on a machine, but for administrative reasons, it is recommended that you assign no more than one to a machine. The following is an example of the layout for a logical host filesystem for Sun Cluster 2.1 with DB2 (See the Sun Cluster 2.1 documentation for instructions on how to add logical host filesystems). The name of the logical host in this example is snap:

/snap/ The logical host filesystem (needed for Sun Cluster 2.1).

/snap/home/
The place to put the high availability instance home directory.

/snap/disks/
The place where SMS filesystems should be placed to make sure that are available after a failover.

Private network
Private networks are used for communicating between two nodes. Heartbeat messages and Remote Procedure Calls (RPCs) travel over these networks to keep the two nodes in synchronous operation, so that they can back up each other in the event of a failover.

Public networks
The public network includes all the primary and logical network interfaces and IP addresses. The logical network interfaces or logical hosts should be referred to when communicating with DB2 on the cluster because these are failed over to the remaining machine at the time of the failover.

Disk group
Disk groups contain one or more shared disks and a list of hosts that can access these disks. Only one host can own the disk groups for exclusive use at a time.

Disk mirroring
It is highly recommended that you mirror disks to increase disk availability. Without mirroring, there is a single point of failure for each disk in the system.

Figure 70 on page 609 shows an example of the components in a cluster.

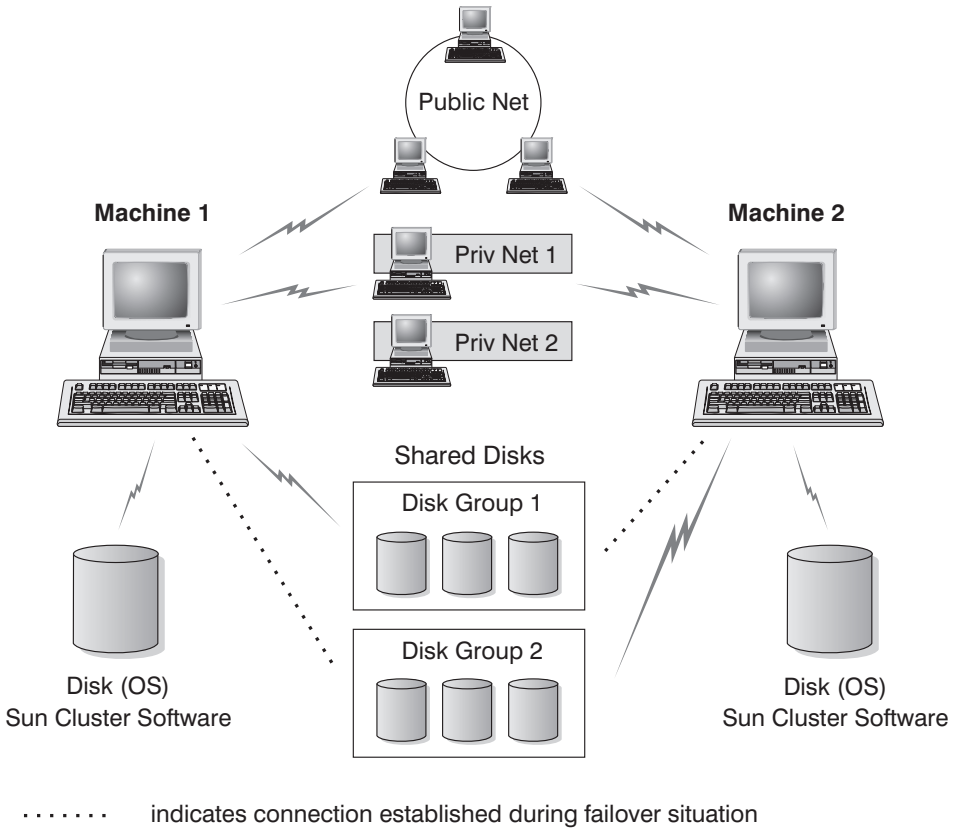


Figure 70. Components in a Cluster

The following sections describe the different types of failover support, and how to implement them.

Failover Configurations

Two types of configuration are available in a DB2 system:

- Hot standby (asymmetric mode)
- Mutual takeover (symmetric mode).

Two modes of failover support are provided. A brief description of each mode and its application to DB2 follows. For each, the simple scenario of a two-server cluster is described.

Hot Standby

One server is being actively used to run DB2, and the second is in

standby mode ready to take over if there is an operating system or hardware failure involving the first server.

Mutual Takeover
Multiple servers can be used to scale to a single database instance using the DB2 Enterprise - Extended Edition product. This is done using a shared-nothing model and partitioning the data such that one or more partitions are running on each server in the cluster. If an operating system or hardware failure occurs on one of the servers, then the other servers will take over the partition (or partitions) of the failing server.

Each of the above configurations can be used to failover one or more partitions of a partitioned database.

Hot Standby Configuration

You can use the hot standby capability to set up failover for a partition or partitions of a partitioned database configuration. If one server fails, then another server in the cluster can substitute for the failed server by automatically transferring the database partitions. To achieve this, the database instance and the actual database must be accessible to both the primary and failover server. This requires that the following installation and configuration tasks be performed:

- The DB2 installation path should be local to each machine and of the same level.
- The DB2 instance path should be on a shared filesystem via HA-NFS.
- The database and the associated containers must be on file systems (or devices) that are accessible to both systems. The disks for the filesystems or devices of a database partition must be in disk groups that are associated with the logical host that hosts the database partition.
- For failover of a partition in a partitioned database configuration, the partition is restarted on the second system: the Sun Cluster 2.1 software modifies the db2nodes.cfg file to point to the failed partition on the new system and starts the partition on that system.
- When a failover occurs, the external communications addresses for supported communication protocols are transparently transferred as part of the failover procedure.

Database Partition Server Failover

Figure 71 on page 611 shows how partitions fail over in a hot standby configuration. System A is running a one or more partitions of the overall configuration and System B is used as the failover system. When System A fails, the partition is restarted on the second system. The failover updates the db2nodes.cfg file, pointing the partition to System B's hostname and netname,

then restarting the partition at the new system. When the failover is complete, all other partitions forward the requests targeted for this partition to System B.

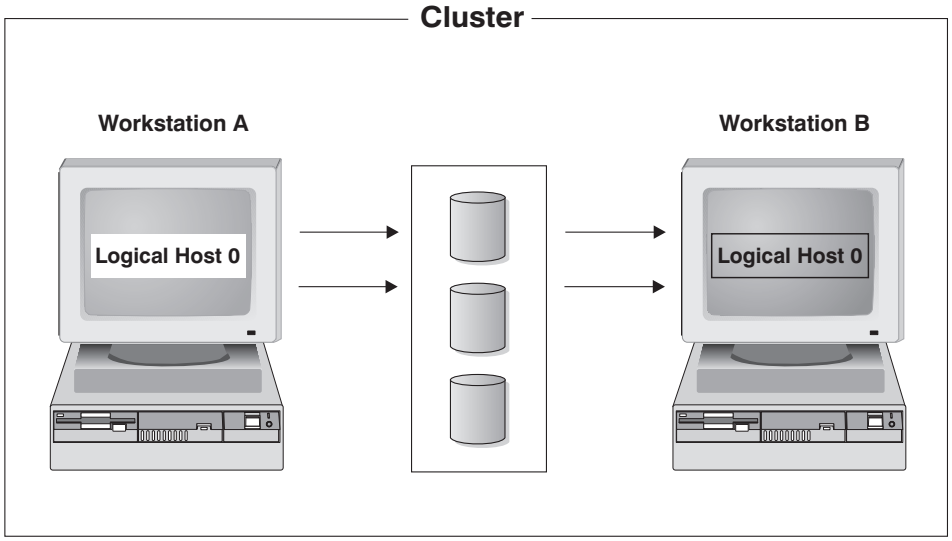


Figure 71. Hot Standby Configuration

The following is a portion of the db2nodes.cfg file before and after the failover. In this example, node numbers 20, 22 and 24 are running on the system named MachineA of the cluster with the netname MachineA-scid0. After the failover, node numbers 20, 22 and 24 are running on the system named MachineB of the cluster and have a netname of MachineB-scid0.

```
Before:
    20 MachineA 0 MachineA-scid0   <= Sun Cluster 2.1
    22 MachineA 1 MachineA-scid0   <= Sun Cluster 2.1
    24 MachineA 2 MachineA-scid0   <= Sun Cluster 2.1

    db2start nodenum 20 restart hostname MachineB port 0 netname MachineB-scid0
    db2start nodenum 22 restart hostname MachineB port 1 netname MachineB-scid0
    db2start nodenum 24 restart hostname MachineB port 2 netname MachineB-scid0

After:
    20 MachineB 0 MachineB-scid0   <= Sun Cluster 2.1
    22 MachineB 1 MachineB-scid0   <= Sun Cluster 2.1
    24 MachineB 2 MachineB-scid0   <= Sun Cluster 2.1
```

Mutual Takeover Configuration

The mutual failover of partitions in a partitioned database environment requires that the failover of the partition occur as a logical node on the

failover server. If two partitions of a partitioned database system run on separate servers of a cluster configured for mutual takeover, the partitions must fail over as logical nodes.

Figure 72 shows an example of a mutual takeover configuration.

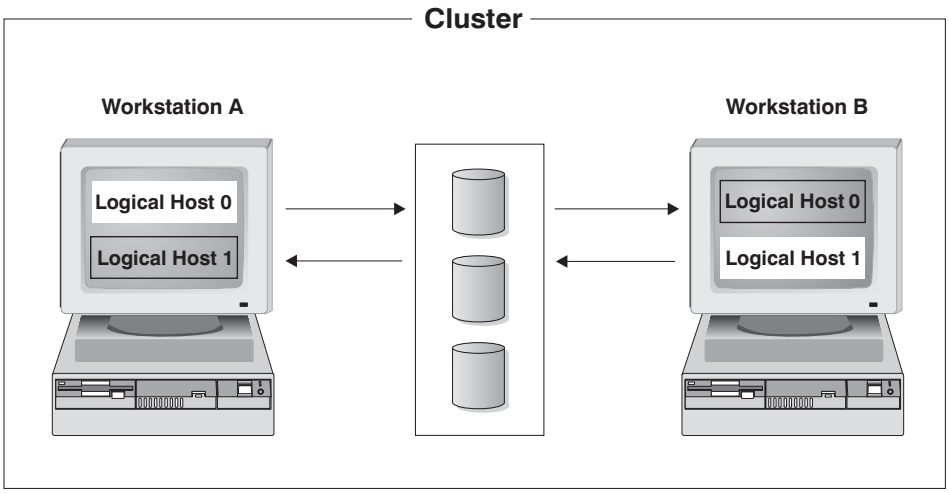


Figure 72. Mutual Takeover Configuration

Another important consideration when configuring a system for mutual partition takeover is the database path of the local partition. When a database is created in a partitioned database environment, it is created on a root path, which is not shared across the database partition 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` file system under `/dbpath/db2inst/NODExxxx`, where `xxxx` represents the node number. After a failover, a database partition will start up on another system with a different `/dbpath` directory. The only filesystems that are moved along with the logical host during a failover are the logical host filesystems. This means that a symbolic link must be created from the logical host file system to the appropriate `/dbpath/db2inst/NODExxxx` path.

For example,

```
cd /dbpath/db2inst
ln -s /log0/disks/db2inst/NODE0001 NODE0001
```

The `hadb2eee_addinst` will set up symbolic links from `INSTHOME/INSTANCE` to the logical host filesystem that corresponds with the various database

partitions (where INSTHOME is the instance owner's home directory, INSTANCE is the instance, and log0 is the logical host that is bound to database partition 1 via the hadb2-eee.cfg file). You must perform this manually for other database directories.

The following example shows a portion of the db2nodes.cfg file before and after the failover. In this example, node numbers 20, 22 and 24 are running on System A which has a hostname of MachineA with a netname of MachineA-scid0. Node numbers 30, 32, and 34 are running on System B which has a hostname of MachineB with a netname of MachineB-scid0. System A in this example is hosting a logical host which is responsible for database partitions 20, 22, and 24. System B is listed as a backup for this logical host and it will host it if System A goes down.

```
Before:
    20 MachineA 0 MachineA-scid0   <= Sun Cluster 2.1
    22 MachineA 1 MachineA-scid0   <= Sun Cluster 2.1
    24 MachineA 2 MachineA-scid0   <= Sun Cluster 2.1
    30 MachineB 0 MachineB-scid0   <= Sun Cluster 2.1
    32 MachineB 1 MachineB-scid0   <= Sun Cluster 2.1
    34 MachineB 2 MachineB-scid0   <= Sun Cluster 2.1

    db2start nodenum 20 restart hostname MachineB port 3 netname MachineB-scid0
    db2start nodenum 22 restart hostname MachineB port 4 netname MachineB-scid0
    db2start nodenum 24 restart hostname MachineB port 5 netname MachineB-scid0

After:
    20 MachineB 3 MachineB-scid0   <= Sun Cluster 2.1
    22 MachineB 4 MachineB-scid0   <= Sun Cluster 2.1
    24 MachineB 5 MachineB-scid0   <= Sun Cluster 2.1
    30 MachineB 0 MachineB-scid0   <= Sun Cluster 2.1
    32 MachineB 1 MachineB-scid0   <= Sun Cluster 2.1
    34 MachineB 2 MachineB-scid0   <= Sun Cluster 2.1
```

If you do decide to use a mutual takeover environment for the coordinator node then you may want to adjust the following database manager configuration parameters:

- *conn_elapse*
- *max_connretries*.

Reducing the value of these parameters will reduce the failover time for the coordinator node, but will increase the risk of an FCM connection timeout. These parameters should be tuned to meet your requirements.

Setting Up Failover Support for a Database System

To set up Sun Cluster 2.1, perform the following steps:

1. Ensure that your system meets the requirements detailed in “Preliminary Requirements” on page 615.
2. Ensure that Sun Cluster 2.1 is installed properly.
3. Create the logical hosts which that will host the database partitions.
4. Create the logical host filesystems, and filesystems for SMS table spaces.
5. Install DB2 on each machine in the cluster (you can use **cconsole** or **ctelnet**, which come with Sun Cluster 2.1).
6. For mutual partition failover, set up HA-NFS either locally or remotely on a separate cluster to export the highly available instance’s home directory.
7. On *one* machine only, create an instance on the HA-NFS filesystem, while ensuring that the user for the instance is created with the same uid on the other machines in the cluster. Also ensure that the groups and services for the instance are also created on the other machines in the cluster.
8. Run the **hadb2eee_addinst** script as root to set up your HA instance or configure the instance manually. The **hadb2eee_addinst** script is provided as an example on how an instance may be set up. The **hadb2eee_addinst** script does the following:
   - Creates the .rhost file in the specified instance owner’s home directory.
   - Creates the db2nodes.cfg file for the instance.
   - Creates the ha-db2eee.cfg file which binds database partitions to a logical host.
   - Sets up symbolic links from the default database path (which is specified by the *dftdbpath* configuration parameter) to the correct logical host filesystem for a database partition.
   - Adds a line for the instance in the /var/db2/v5/db2tabeee file.
   - Tries to run non-interactively on the other machines in the cluster.
9. Run **hadb2start** to start the high availability environment

Choosing a Failover Configuration

To choose a failover configuration, perform the following steps:

1. Set up the machines to use either a hot standby or mutual takeover configuration. For a hot standby configuration, use one logical host. For a mutual takeover, use two or more logical hosts.
2. Decide on the amount of disk space that is required for each logical host and its resources, such as raw devices or SMS table space containers. Whether you use SMS or DMS (raw devices) table spaces, any disks belonging to a logical host must be included in its disk groups.

   Table space considerations

   You must decide on the type of table space that you want to use. If you want to use SMS table spaces, you must set them up using disks from the disk groups that belong to a logical host. In

addition, you must include the filesystem in the vfstab for the logical host. Refer to the Sun Cluster 2.1 documentation for information about how to add a file system to a logical host.

There are benefits and costs associated with using either SMS or DMS table spaces. For example, SMS table spaces reside on file systems that must be file-system checked before they are mounted. This can add a considerable amount of overhead when failover occurs, and can result in the Sun Cluster 2.1 software timing out. If you use SMS table spaces, ensure that they are journaled files systems, which require less time to check after a failover.

DMS table spaces do not have to be file-system checked during failover, which can reduce the failover time for the high availability scripts.

You should remember that, for both SMS and DMS table spaces, committed transactions that are written to the logs will be applied to the database during crash recovery after the database server fails over.

Preliminary Requirements

Any logical host that you want in a cluster must have the following directories available:

/LOGICAL_HOST
Is the name of the logical host that runs the partition

/LOGICAL_HOST/home
Is where the home directories reside

/LOGICAL_HOST/disks
Is where the SMS table spaces reside for the database partitions

For example:

```
/log0
/log0/home/db2eee
/log0/disks/db2eee
```

Where `log0` is the logical host and `db2eee` is the highly available instance.

Scripts and Programs

All of the following scripts are in the directory /opt/IBMdb2/V5.0/ha/UDB-EEE_SC2.x/bin. Included are:

hadb2eee_addinst
The sample instance setup script

hadb2eee_reg
Registers DB2 Enterprise - Extended Edition for high availability

hadb2eee_startnet
Script that starts partitions for a logical host. This script is run automatically by Sun Cluster 2.1. You should not run this script manually.

hadb2eee_stopnet
Script that stops partitions for a logical host. This script is run automatically by Sun Cluster 2.1. You should not run this script manually.

hadb2eee_unreg
Unregisters DB2 Enterprise - Extended Edition for high availability

hadb2stat
Shows the current status of DB2 Enterprise - Extended Edition

hadb2start
Starts DB2 in the highly available environment.

hadb2stop
Stops DB2 in the highly available environment.

The hadb2eee_startnet and hadb2eee_stopnet scripts are used during a failover. The hadb2eee_startnet script starts partitions on a physical machine, while hadb2eee_stopnet stops partitions on a physical machine. Both the start and stop scripts read the /var/db2/v5/db2tabeee configuration file to find out which DB2 instances are highly available. See “Enabling Failover for an Instance” on page 617 for information about this file.

Creating a DB2 Instance

Refer to the *DB2 Enterprise - Extended Edition for UNIX Quick Beginnings* for information about creating an instance.

Registering the DB2 Resource with Sun Cluster 2.1

If you are using local HA-NFS for your cluster, you must register and set up HA-NFS before HA DB2 EEE. The hadb2eee_reg script may look something like this:

```
# Make sure you register hanfs with every logical node even though only
# one will use it. This is to fix a dependency issue with SC2.x.
#

TIMEOUT=600
STARTUP=1
#DEPONNFS=
DEPONNFS="-d nfs"
```

```
# Register HA-NFS
#
#hareg -s -r nfs
#hareg -y nfs

hareg -r hadb2eee -b /opt/IBMdb2/V5.0/ha/UDB-EEE_SC2.x/bin/ -m START=hadb2eee_st
art,START_NET=hadb2eee_startnet,STOP_NET=hadb2eee_stopnet,ABORT_NET=hadb2eee_abo
rtnet -t START_NET=$TIMEOUT,STOP_NET=$TIMEOUT $DEPONNFS

if [[ STARTUP -eq 1 ]]
then
  hareg -y hadb2eee
fi
```

Where:

TIMEOUT
Is the timeout for the Sun Cluster agent to start and stop DB2.

STARTUP
Specifies whether to start the high availability environment after registering HA DB2 EEE.

DEPONNFS
Set this to an empty string if you are using a remote HA-NFS server. If you are using a local HA-NFS server, ensure that this is set to `-d nfs`, and that the lines that register HA-NFS are uncommented.

Enabling Failover for an Instance

To enable an instance for failover, you create an entry for it in the /var/db2/v5/db2tabeee file. This file must be kept consistent across all the machines in the cluster. Entries in this file are in the form:

```
TYPE INSTANCE NFS_HOST ON HA-NFS_DIR LOCAL_MOUNT_POINT
```

Where:

TYPE Is the type of instance. The value can be one of the following:

- DATA to indicate a database instance.
- ADMIN to indicate an administration server instance.

INSTANCE
Is the user name of the instance owner.

NFS_HOST
Is the logical host which is hosting the HA-NFS filesystem.

ON/OFF
Specifies whether the instance is highly available (ON) or not (OFF).

HA-NFS_DIR
The directory on the HA-NFS host to mount.

LOCAL_MOUNT_POINT
The local mount point for the HA-NFS.

An example might be:

```
DATA db2eee sphere ON /log0/home /export/ha_home
```

In this example, the instance owner's home directory should be placed under /export/ha_home.

Binding Database Partition Servers to a Logical Host

You use the file called $INSTHOME/sqllib/hadb2-eee.cfg to *bind* database partitions to a logical host. Bind, in this context, means that the file ensures that the partitions follow the logical hosts around the cluster, starting on the machine in the cluster that hosts the logical host. Entries in this file are in the form:

```
NODE: log0  0
NODE: log0 10
NODE: log0 12
NODE: log1 33
NODE: log1 45
NODE: log1 59
```

In this example, logical host log0 is responsible for partitions 0, 10, and 12, while logical host log1 is responsible for partitions 33, 45, and 59. These logical hosts are responsible for both starting and stopping the partitions during a failover situation.

Note: There must be a one-to-one relationship between the partitions in this file and the db2nodes.cfg file.

How Failover Processing Works

When a failover occurs, the hadb2eee_startnet and hadb2eee_stopnet programs read the /var/db2/v5/db2tabeee file to find out which DB2 instances are highly available. Then for each highly available instance, they read the configuration file $INSTHOME/sqllib/hadb2-eee.cfg, which binds partitions to logical hosts.

Information about the failover process is sent to the syslog using the facility set to LOG_USER and the priority set to LOG_ERR.

Setting Up a Hot Standby Configuration

To set up a hot standby configuration, bind all of the partitions to one logical host that is hosted by one of the servers in the cluster. When you finish, the $INSTHOME/sqllib/hadb2-eee.cfg file should resemble the following:

```
NODE: log0  0
NODE: log0 10
NODE: log0 12
NODE: log0 33
NODE: log0 45
NODE: log0 59
```

If the logical host `log0` fails over, all the database partitions associated with it will fail over as well.

Setting Up a Mutual Takeover Configuration

To set up a mutual takeover configuration, bind the partitions to two or more logical hosts. When you finish, the $INSTHOME/sqllib/hadb2-eee.cfg file should resemble the following:

```
NODE: log0  0
NODE: log0 10
NODE: log0 12
NODE: log0 33
NODE: log1 45
NODE: log1 59
```

You do not need to set up a completely symmetric configuration. As the example shows, the logical host `log0` supports more partitions than the logical host `log1` (partitions `0`, `10`, `12` and `33` for logical host `log0` versus partitions `45` and `59` for logical host `log1`). Because you do not have to implement a symmetric configuration, a mutual takeover configuration provides an amount of flexibility that will support any situation.

Starting and Stopping DB2

To start DB2 in a failover environment, use the **hadb2start** command. This command both enables the failover environment, and starts DB2.

If you want to stop DB2, use the **hadb2stop** command. This command both disables the failover environment and stops DB2.

Note: If you do not issue **hadb2stop** and you use **db2stop**, Sun Cluster 2.1 may assume that the DB2 instance needs to be failed over.

Running Scripts During a Failover

The **/var/db2/v5/failover.eee** script runs automatically when a failover occurs. You can use this script to send email (for example, to notify support staff) of the failover situation. You should keep the commands in this script to a minimum, because it runs before DB2 is started. Depending on whether DB2 is starting or stopping, the following scripts will also run (if they are available) for each instance.

Note: You must create the *$INSTHOME*/sqllib/ha directory and create these scripts to be executables. You should ensure that you have backup copies of these scripts.

- *$INSTHOME***/sqllib/ha/pre_db2starteee**

  This file takes as an argument the number of logical hosts that are currently running on the failover machine. If this script exists, it runs immediately before the **db2start** command.
- *$INSTHOME***/sqllib/ha/pre_db2stopeee**

  This file takes as an argument the number of logical hosts that are currently running on the failover machine. If this script exists, it runs immediately before the **db2stop** command.

  Note: This script may not be run if the machine crashes.
- *$INSTHOME***/sqllib/ha/post_failovereee**

  This file runs just after a failover and is used to such tasks as restart databases.

Considerations for Table Spaces

You must decide on the type of table space that you want to use. If you want to use SMS table spaces, you must set them up using disks from the disk groups that belong to a logical host. In addition, you must include them in the vfstab for the logical host. Refer to the Sun Cluster 2.1 documentation for information about how to add a file system to a logical host.

There are benefits and costs associated with using either SMS or DMS table spaces. For example, SMS table spaces reside on file systems that must be file-system checked before they are mounted. This can add a considerable amount of overhead when failover occurs, and can result in the Sun Cluster 2.1 software timing out. If you use SMS table spaces, ensure that they are journaled file systems, which require less time to check after a failover.

DMS table spaces that use raw devices do not have to be file-system checked during failover. This can reduce the failover time for the high availability

scripts, but you should remember that committed transactions that are written to the logs will be applied to the database during crash recovery after the database server fails over.

If you are using raw devices for table spaces (that is, you are using DMS table spaces), you must ensure that the disks are part of the disk group of the logical host.

Client Application Considerations

Client applications should communicate with the high availability services only through the logical hostname of the logical host of the high availability service. You should ensure that client applications are written to accept a communications error and possibly retry after a few minutes.

Consider a typical client connection. The client is connected to machineA through the logical host called snap. If machineA fails, then snap fails over to machineB. According to machineB, the client connection does not exist, and will send the client a connect reset message, which will appear to the client as a communication error. The client must reconnect to the server to obtain a new connection from machineB when DB2 starts.

Part 5. Appendixes

Appendix A. How the DB2 Library Is Structured

The DB2 Universal Database library consists of SmartGuides, online help, books and sample programs in HTML format. This section describes the information that is provided, and how to access it.

To access product information online, you can use the Information Center. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. See "Accessing Information with the Information Center" on page 636 for details.

Completing Tasks with SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available through the Control Center and the Client Configuration Assistant. The following table lists the SmartGuides.

Note: Create Database, Index, and Configure Multisite Update SmartGuide are available for the partitioned database environment.

SmartGuide	Helps You to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click **Add**.
Back up Database	Determine, create, and schedule a backup plan.	From the Control Center, click with the right mouse button on the database you want to back up and select **Backup**->**Database using SmartGuide**.
Configure Multisite Update SmartGuide	Perform a multi-site update, a distributed transaction, or a two-phase commit.	From the Control Center, click with the right mouse button on the **Database** icon and select **Multisite Update**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**Database using SmartGuide**.

SmartGuide	Helps You to...	How to Access...
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, click with the right mouse button on the **Tables** icon and select **Create->Table using SmartGuide**.
Create Table Space	Create a new table space.	From the Control Center, click with the right mouse button on the **Table spaces** icon and select **Create->Table space using SmartGuide**.
Index	Advise which indexes to create and drop for all your queries.	From the Control Center, click with the right mouse button on the **Index** icon and select **Create->Index using SmartGuide**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, click with the right mouse button on the database you want to tune and select **Configure using SmartGuide**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, click with the right mouse button on the database you want to restore and select **Restore->Database using SmartGuide**.

Accessing Online Help

Online help is available with all DB2 components. The following table describes the various types of help. You can also access DB2 information through the Information Center. For information see "Accessing Information with the Information Center" on page 636.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: ? *command* where *command* is a keyword or the entire command. For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.

Type of Help	Contents	How to Access...
Control Center Help *Client Configuration Assistant Help* *Event Analyzer Help* *Command Center Help*	Explains the tasks you can perform in a window or notebook. The help includes prerequisite information you need to know, and describes how to use the window or notebook controls.	From a window or notebook, click the **Help** push button or press the F1 key.

Message Help	Describes the cause of a message, and any action you should take.	From the command line processor in interactive mode, enter:
		
		`? XXXnnnnn`
		
		where *XXXnnnnn* is a valid message identifier.
		
		For example, `? SQL30081` displays help about the SQL30081 message.
		
		To view message help one screen at a time, enter:
		
		`? XXXnnnnn | more`
		
		To save message help in a file, enter:
		
		`? XXXnnnnn > filename.ext`
		
		where *filename.ext* is the file where you want to save the message help.

SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: `help statement` where *statement* is an SQL statement. For example, **help** SELECT displays help about the SELECT statement. **Note:** SQL help is not available on UNIX-based platforms.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: **?** *sqlstate* or **?** *class-code* where *sqlstate* is a valid five-digit SQL state and *class-code* is the first two digits of the SQL state. For example, `? 08003` displays help for the 08003 SQL state, while `? 08` displays help for the 08 class code.

DB2 Information – Hardcopy and Online

The table in this section lists the DB2 books. They are divided into two groups:

Cross-platform books
These books contain the common DB2 information for all platforms.

Platform-specific books
These books are for DB2 on a specific platform. For example, there are separate *Quick Beginnings* books for DB2 on OS/2, on Windows NT, and on the UNIX-based platforms.

Cross-platform sample programs in HTML
These samples are the HTML version of the sample programs that are installed with the SDK. They are for informational purposes and do not replace the actual programs.

Most books are available in HTML and PostScript format, or you can choose to order a hardcopy from IBM. The exceptions are noted in the table.

On OS/2 and Windows platforms, HTML documentation files can be installed under the doc\html subdirectory. Depending on the language of your system, some files may be in that language, and the remainder are in English.

On UNIX platforms, you can install multiple language versions of the HTML documentation files under the doc/%L/html subdirectories. Any documentation that is not available in a national language is shown in English.

You can obtain DB2 books and access information in a variety of different ways:

View See "Viewing Online Information" on page 635.

Search See "Searching Online Information" on page 638.

Print See "Printing the PostScript Books" on page 638.

Order See "Ordering the Printed Books" on page 639.

Name	Description	Form Number File Name for Online Book	HTML Directory
	Cross-Platform Books		

Name	Description	Form Number File Name for Online Book	HTML Directory
Administration Guide	*Administration Guide, Design and Implementation* contains information required to design, implement, and maintain a database. It also describes database access using the Control Center(whether local or in a client/server environment), auditing, database recovery, distributed database support, and high availability. *Administration Guide, Performance* contains information that focuses on the database environment, such as application performance evaluation and tuning. You can order both volumes of the *Administration Guide* in the English language in North America using the form number SBOF-8922.	Volume 1 SC09-2839 db2d1x60 Volume 2 SC09-2840 db2d2x60	db2d0
Administrative API Reference	Describes the DB2 application programming interfaces (APIs) and data structures you can use to manage your databases. Explains how to call APIs from your applications.	SC09-2841 db2b0x60	db2b0
Application Building Guide	Provides environment setup information and step-by-step instructions about how to compile, link, and run DB2 applications on Windows, OS/2, and UNIX-based platforms. This book combines the *Building Applications* books for the OS/2, Windows, and UNIX-based environments.	SC09-2842 db2axx60	db2ax
APPC, CPI-C and SNA Sense Codes	Provides general information about APPC, CPI-C, and SNA sense codes that you may encounter when using DB2 Universal Database products. **Note:** Available in HTML format only.	No form number db2apx60	db2ap

Name	Description	Form Number File Name for Online Book	HTML Directory
Application Development Guide	Explains how to develop applications that access DB2 databases using embedded SQL or JDBC, how to write stored procedures, user-defined types, user-defined functions, and how to use triggers. It also discusses programming techniques and performance considerations. This book was formerly known as the *Embedded SQL Programming Guide*.	SC09-2845 db2a0x60	db2a0
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	SC09-2843 db2l0x60	db2l0
Command Reference	Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database.	SC09-2844 db2n0x60	db2n0
Data Movement Utilities Guide and Reference	Explains how to use the Load, Import, Export, Autoloader, and Data Propogation utilities to work with the data in the database.	SC09-2858 db2dmx60	db2dm
DB2 Connect Personal Edition Quick Beginnings	Provides planning, installing, and configuring information for DB2 Connect Personal Edition.	GC09-2830 db2c1x60	db2c1
DB2 Connect User's Guide	Provides concepts, programming and general usage information about the DB2 Connect products.	SC09-2838 db2c0x60	db2c0
Connectivity Supplement	Provides setup and reference information on how to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA application requesters with DB2 Universal Database servers, and on how to use DRDA application servers with DB2 Connect application requesters. **Note:** Available in HTML and PostScript formats only.	No form number db2h1x60	db2h1
Glossary	Provides a comprehensive list of all DB2 terms and definitions. **Note:** Available in HTML format only.	No form number db2t0x50	db2t0

Name	Description	Form Number File Name for Online Book	HTML Directory
Installation and Configuration Supplement	Guides you through the planning, installation, and set up of platform-specific DB2 clients. This supplement contains information on binding, setting up client and server communications, DB2 GUI tools, DRDA AS, distributed installation, and the configuration of distributed requests and access methods to heterogeneous data sources.	GC09-2857 db2iyx60	db2iy
Message Reference	Lists messages and codes issued by DB2, and describes the actions you should take.	GC09-2846 db2m0x60	db2m0
Replication Guide and Reference	Provides planning, configuration, administration, and usage information for the IBM Replication tools supplied with DB2.	SC26-9642 db2e0x60	db2e0
SQL Getting Started	Introduces SQL concepts, and provides examples for many constructs and tasks.	SC09-2856 db2y0x60	db2y0
SQL Reference, Volume 1 and Volume 2	Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views. You can order both volumes of the *SQL Reference* in the English language in North America with the form number SBOF-8923.	SBOF-8923 Volume 1 db2s1x60 Volume 2 db2s2x60	db2s0
System Monitor Guide and Reference	Describes how to collect different kinds of information about databases and the database manager. Explains how to use the information to understand database activity, improve performance, and determine the cause of problems.	SC09-2849 db2f0x60	db2f0
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	S10J-8169	db2p0

Name	Description	Form Number File Name for Online Book	HTML Directory
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 6.0, including information about Java-based tools.	SC09-2851 db2q0x60	db2q0
	Platform-Specific Books		
Administering Satellites Guide and Reference	Provides planning, configuration, administration, and usage information for satellites.	GC09-2821 db2dsx60	db2ds
DB2 Personal Edition Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on the OS/2, Windows 95, and Windows NT operating systems.	GC09-2831 db2i1x60	db2i1
DB2 for OS/2 Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the OS/2 operating system. Also contains installing and setup information for many supported clients.	GC09-2834 db2i2x60	db2i2
DB2 for UNIX Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for many supported clients.	GC09-2836 db2ixx60	db2ix
DB2 for Windows NT Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for many supported clients.	GC09-2835 db2i6x60	db2i6
DB2 Enterprise - Extended Edition for UNIX Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for UNIX. Also contains installing and setup information for many supported clients.	GC09-2832 db2v3x60	db2v3

Name	Description	Form Number File Name for Online Book	HTML Directory
DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for Windows NT. Also contains installing and setup information for many supported clients.	GC09-2833 db2v6x60	db2v6
DB2 Connect Enterprise Edition for OS/2 and Windows NT Quick Beginnings	Provides planning, migration, installation, and configuration information for DB2 Connect Enterprise Edition on the OS/2 and Windows NT operating systems. Also contains installation and setup information for many supported clients. This book was formerly part of the *DB2 Connect Enterprise Edition Quick Beginnings.*	GC09-2828 db2c6x60	db2c6
DB2 Connect Enterprise Edition for UNIX Quick Beginnings	Provides planning, migration, installation, configuration, and usage information for DB2 Connect Enterprise Edition in UNIX-based platforms. Also contains installation and setup information for many supported clients. This book was formerly part of the *DB2 Connect Enterprise Edition Quick Beginnings.*	GC09-2829 db2cyx60	db2cy
DB2 Data Links Manager for AIX Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for AIX.	GC09-2837 db2z0x60	db2z0
DB2 Data Links Manager for Windows NT Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for Windows NT.	GC09-2827 db2z6x60	db2z6
DB2 Query Patroller Administration Guide	Provides administration information on DB2 Query Patrol.	SC09-2859 db2dwx60	db2dw
DB2 Query Patroller Installation Guide	Provides installation information on DB2 Query Patrol.	GC09-2860 db2iwx60	db2iw
DB2 Query Patroller User's Guide	Describes how to use the tools and functions of the DB2 Query Patrol.	SC09-2861 db2wwx60	db2ww

Name	Description	Form Number File Name for Online Book	HTML Directory
	Cross-Platform Sample Programs in HTML		
Sample programs in HTML	Provides the sample programs in HTML format for the programming languages on all platforms supported by DB2 for informational purposes (not all samples are available in all languages). Only available when the SDK is installed. See *Application Building Guide* for more information on the actual programs. **Note:** Available in HTML format only.	No form number	db2hs/c db2hs/cli db2hs/clp db2hs/cpp db2hs/cobol db2hs/cobol_mf db2hs/fortran db2hs/java db2hs/rexx

Notes:

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e60` indicates that the *Administration Guide* is in English. The following letters are used in the file names to indicate the language of a book:

Language	Identifier
Brazilian Portuguese	b
Bulgarian	u
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Russian	r
Simp. Chinese	c
Slovenian	l
Spanish	z

Swedish	s
Trad. Chinese	t
Turkish	m

2. For late breaking information that could not be included in the DB2 books:
   - On UNIX-based platforms, see the Release.Notes file. This file is located in the DB2DIR/Readme/%L directory, where %L is the locale name and DB2DIR is:
     - /usr/lpp/db2_06_01 on AIX
     - /opt/IBMdb2/V6.1 on HP-UX, Solaris, SCO UnixWare 7, and Silicon Graphics IRIX
     - /usr/IBMdb2/V6.1 on Linux.
   - On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed.
   - Under Windows Start menu

Viewing Online Information

The manuals included with this product are in Hypertext Markup Language (HTML) softcopy format. Softcopy format enables you to search or browse the information, and provides hypertext links to related information. It also makes it easier to share the library across your site.

You can view the online books or sample programs with any browser that conforms to HTML Version 3.2 specifications.

To view online books or sample programs on all platforms other than SCO UnixWare 7:

- If you are running DB2 administration tools, use the Information Center. See "Accessing Information with the Information Center" on page 636 for details.
- Select the Open Page menu item of your Web browser. The page you open contains descriptions of and links to DB2 information:
  - On UNIX-based platforms, open the following page:

    ```
    file:/INSTHOME/sqllib/doc/%L/html/index.htm
    ```

    where *%L* is the locale name.
  - On other platforms, open the following page:

    ```
    sqllib\doc\html\index.htm
    ```

    The path is located on the drive where DB2 is installed.

If you have not installed the Information Center, you can open the page by double-clicking on the **DB2 Online Books** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

To view online books or sample programs on the SCO UnixWare 7:

- DB2 Universal Database for SCO UnixWare 7 uses the native SCOhelp utility to search the DB2 information. You can access SCOhelp by the following methods:
  - entering the ″scohelp″ command on the command line,
  - selecting the `Help` menu in the Control Panel of the CDE desktop or
  - selecting `Help` in the Root menu of the Panorama desktop

  For more information on SCOhelp, refer to the *Installation and Configuration Supplement*.

Accessing Information with the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on all platforms on which the DB2 administration tools are available.

Depending on your system, you can access the Information Center from the:

- Main product folder
- Toolbar in the Control Center
- Windows Start menu
- Help menu of the Control Center

The Information Center provides the following kinds of information. Click the appropriate tab to look at the information:

Tasks	Lists tasks you can perform using DB2.
Reference	Lists DB2 reference information, such as keywords, commands, and APIs.
Books	Lists DB2 books.
Troubleshooting	Lists categories of error messages and their recovery actions.
Sample Programs	Lists sample programs that come with the DB2 Software Developer's Kit. If the Software Developer's Kit is not installed, this tab is not displayed.
Web	Lists DB2 information on the World Wide

Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides some search capabilities, so you can look for specific topics, and filter capabilities to limit the scope of your searches.

For a full text search, click the Search button of the Information Center follow the *Search DB2 Books* link in each HTML file.

The HTML search server is usually started automatically. If a search in the HTML information does not work, you may have to start the search server by double-clicking its icon on the Windows or OS/2 desktop.

Refer to the release notes if you experience any other problems when searching the HTML information.

Note: Search function is not available in the Linux and Silicon Graphics environments.

Setting Up a Document Server

By default, the DB2 information is installed on your local system. This means that each person who needs access to the DB2 information must install the same files. To have the DB2 information stored in a single location, use the following instructions:

1. Copy all files and subdirectories from \sqllib\doc\html on your local system to a Web server. Each book has its own subdirectory containing all the necessary HTML and GIF files that make up the book. Ensure that the directory structure remains the same.
2. Configure the Web server to look for the files in the new location. For information, see the NetQuestion Appendix in *Installation and Configuration Supplement*.
3. If you are using the Java version of the Information Center, you can specify a base URL for all HTML files. You should use the URL for the list of books.
4. Once you are able to view the book files, you should bookmark commonly viewed topics. Among those, you will probably want to bookmark the following pages:

- List of books
- Tables of contents of frequently used books
- Frequently referenced articles, such as the *ALTER TABLE* topic
- The Search form

For information about setting up a search, see the NetQuestion Appendix in *Installation and Configuration Supplement* book.

Searching Online Information

To search for information in the HTML books, you can do the following:

- Click on **Search the DB2 Books** at the bottom of any page in the HTML books. Use the search form to find a specific topic. This function is not available in the Linux or Silicon Graphics IRIX environments.
- Click on **Index** at the bottom of any page in an HTML book. Use the index to find a specific topic in the book.
- Display the table of contents or index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. See "Accessing Information with the Information Center" on page 636 for details.

Printing the PostScript Books

If you prefer to have printed copies of the manuals, you can decompress and print PostScript versions. For the file name of each book in the library, see the table in "DB2 Information – Hardcopy and Online" on page 628. Specify the full path name for the file you intend to print.

On OS/2 and Windows platforms:

1. Copy the compressed PostScript files to a hard drive on your system. The files have a file extension of .exe and are located in the x:\doc*language*\books\ps directory, where x: is the letter representing the CD-ROM drive and *language* is the two-character country code that represents your language (for example, EN for English).
2. Decompress the file that corresponds to the book that you want. Each compressed book is a self-extracting executable file. To decompress the

book, simply run it as you would run any other executable program. The result from this step is a printable PostScript file with a file extension of .ps.

3. Ensure that your default printer is a PostScript printer capable of printing Level 1 (or equivalent) files.
4. Enter the following command from a command line:

   ```
   print filename.ps
   ```

On UNIX-based platforms:

1. Mount the CD-ROM. Refer to your *Quick Beginnings* manual for the procedures to mount the CD-ROM.
2. Change to /cdrom/doc/%L/ps directory on the CD-ROM, where */cdrom* is the mount point of the CD-ROM and *%L* is the name of the desired locale. The manuals will be installed in the previously-mentioned directory with file names ending with .ps.Z.
3. Decompress and print the manual you require using the following command:
   - For AIX:

     ```
     zcat filename | qprt -P PSPrinter_queue
     ```

   - For HP-UX, Solaris, or SCO UnixWare 7:

     ```
     zcat filename | lp -d PSPrinter_queue
     ```

   - For Linux:

     ```
     zcat filename | lpr -P PSPrinter_queue
     ```

   - For Silicon Graphics IRIX:

     ```
     zcat < filename | lp -d PSPrinter_queue
     ```

   where *filename* is the full path name and extension of the compressed PostScript file and *PSprinter_queue* is the name of the PostScript printer queue.

   For example, to print the English version of *DB2 for UNIX Quick Beginnings* on AIX, you can use the following command:

   ```
   zcat /cdrom/doc/en/ps/db2ixe60.ps.Z | qprt -P ps1
   ```

Ordering the Printed Books

You can order the printed DB2 manuals either as a set or individually. There are two sets of books available. The form number for the entire set of DB2 books is SBOF-8926-00. The form number for the books listed under the heading "Cross-Platform Books" is SBOF-8924-00.

Note: These form numbers only apply if you are ordering books that are printed in the English language in North America.

You can also order books individually by the form number listed in “DB2 Information – Hardcopy and Online” on page 628. To order printed versions, contact your IBM authorized dealer or marketing representative, or phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada.

Appendix B. Planning Database Migration

When you migrate your database, the following events occur, as necessary, depending on which level of DB2 you are migrating from:

- The following database entities are migrated:
  - Database configuration file
  - Database system catalog tables
  - Database directories
  - Database log file header
- The database is relocated to a new database path (only applicable to a migration from DB2 Version 2.x).
- 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 buffer pool file is created in the database directory (only applicable to a migration from DB2 Version 2.x).
- 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 specific operations are 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 V6, 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 V6:

- Migration is only supported from V2.x or V5.x. Migration from DB2 V1.2 Parallel Edition is not supported; only DB2 UDB EEE V5.x is supported. Earlier versions of DB2 (Database Manager) must be migrated to V2.x or V5.x before attempting to migrate to V6.
- Issuing the migration command from a V6 client to migrate a database on a V6 Server is supported. However, issuing a migration command from earlier versions of DB2 clients to a V6 Server is not supported.
- Migration between platforms is not supported.
- User objects within your database cannot have V6 reserved schema names as object qualifiers. These reserved schema names include: SYSCAT, SYSSTAT, and SYSFUN.
- Database objects in DB2 Version 2.x 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.
- User-defined distinct types using the names BIGINT, REAL, DATALINK, or REFERENCE must be renamed before migrating the database.
- 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 (V2.x or V5.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.

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:

Configuration Parameters

When migrating from Version 2.x to Version 6, a small number of parameters are not preserved due to a change in the behavior of the associated heap.

In order to take advantage of Version 6 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 Version 2.x 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.

You cannot migrate a database where there are user-defined distinct types using BIGINT, REAL, DATALINK, or REFERENCE as the name of the type.

You cannot migrate a database where there are table spaces with containers which were defined using an absolute database path.

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.
   - Redefine containers so that absolute database paths are not used.
   - Correct database connection states.
   - Remove all dependencies from objects on scalar function SYSFUN.DIFFERENCE.
   - When a structured type and a user-defined function both live in the same schema, have identical names, and the function is a zero-argument function, there can be problems. As an example, migration might attempt to create a zero-argument constrictor function for the structured type that returns an initialized instance of the type. The result would be that this function would clash with the existing user-defined function (UDF) of the same name.
   - Drop and recreate objects using invalid user-defined distinct types, named BIGINT, REAL, DATALINK, or REFERENCE.

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 6. 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 V6 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 V6 is complete are not recoverable. That is, if sometime following the completion of the installation and migration to V6, the database needs to be restored (to a V6 level), the logs from before V6 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.

   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 V6, 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 6. (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 6 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 6 semantics. This also includes creating the indexes with bi-directional pointers which are of benefit when scanning through leaf nodes to retrieve ascending or descending ranges of values. 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 6 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. DB2RBIND has an argument called “all” which, when specified, rebinds all valid and invalid packages. 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 6 enhancements.
10. Optionally, migrate Explain tables if you have been using the Explain tables and are planning to use them in Version 6. There are several new columns in the tables. Refer to the “SQL Explain Facility” and the “Explain Tables and Definitions” in *Administration Guide, Performance* for more information. The *Quick Beginnings* manuals have details on migrating Version 2.x and Version 5.x Explain tables to Version 6.

Complete details on the migration steps are found in the *Quick Beginnings* manuals for your platform.

Appendix C. 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 is used in an existing application, it will either 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:

- System Catalog Views
- Application Programming
- SQL
- Database Security and Tuning
- Utilities and Tools
- Connectivity and Coexistence
- Configuration Parameters.

These categories are found in each of two major sections:

- DB2 Universal Database Version 6 Incompatibilities
- DB2 Universal Database Version 5 Incompatibilities

Each incompatibility includes a description of the change in that particular version 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:

OS/2 OS/2

UNIX Unix-based operating systems supported by DB2

WIN Microsoft Windows platforms supported by DB2

Extended
DB2 Extended Edition, and/or Enterprise – Extended Edition

DB2 PE
DB2 Parallel Edition, Version 1.2 (shown only for previous version consistency)

DB2 Universal Database Planned Incompatibilities

This section focuses on the DB2 Universal Database incompatibilities that we are reserving the right to do two versions in the future, following Version 6. Users of DB2 Universal Database are warned to code applications with these in mind, as well as to modify existing applications to make migration to future versions easier.

Read-only Views in a Future Version of DB2 Universal Database

UNIX	OS/2	WIN	Extended

Change

The system catalog views will be read-only views. The SYSSTAT views will continue to be updateable.

Symptom

UPDATE statements which used to work against columns in the SYSCAT views will fail.

Explanation

Tools or applications are coded to change values in the catalog by updating the column as defined in the SYSCAT view.

Resolution

Change the tool or application to change the catalog by updating the column as defined in the SYSSTAT view.

PK_COLNAMES and FK_COLNAMES in a Future Version of DB2 Universal Database

UNIX	OS/2	WIN	Extended

Change

The SYSCAT.REFERENCES columns PK_COLNAMES and FK_COLNAMES will no longer be available.

Symptom

Column does not exist and an error is returned.

Explanation

Tools or applications are coded to use the obsolete PK_COLNAMES and FK_COLNAMES columns.

Resolution

Change the tool or application to use the SYSCAT.KEYCOLUSE view instead.

COLNAMES No Longer Available in a Future Version of DB2 Universal Database

UNIX	OS/2	WIN	Extended

Change

The SYSCAT.INDEXES column COLNAMES will no longer be available.

Symptom

Column does not exist and an error is returned.

Explanation

Tools or applications are coded to use the obsolete COLNAMES column.

Resolution

Change the tool or application to use the SYSCAT.INDEXCOLUSE view instead.

DB2 Universal Database Version 6 Incompatibilities

This section focuses on the DB2 Universal Database Version 6 incompatibilities.

System Catalog Views

System Catalog Views in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

In the system catalog views, new codes have been introduced: “U” for typed tables, and “W” for typed views.

Symptom:

Queries that search for tables and views in the system catalogs, using the typecode “T” for tables and “V” for views, will no longer find typed tables and views.

Explanation:

Several system catalogs, including the system catalog views named TABLES, PACKAGEDEP, TRIGDEP, and VIEWDEP, have a column named TYPE or BTYPE containing a one-letter typecode. In Version 5.2, the typecode “T” was used for all tables, and “V” was used for all views. In Version 6, untyped tables will continue to have a typecode of “T” and typed tables will have a new typecode of “U”. Similarly, untyped views will continue to have a typecode of “V” and typed views will have a new typecode of “W”. Also, a new kind of table called a hierarchy table, not directly created by users but used by the system to implement table hierarchies, will appear in the system catalog tables with a typecode of “H”.

Resolution:

Change the tool or application to recognize the codes for typed tables and views. If the tool or application needs a logical view of tables, then typecodes “T”, “U”, “V”, and “W” should be used. If the tool or application needs a physical view of tables, including hierarchy tables, then typecodes “T” and “H” should be used.

Primary and Foreign Key Column Names in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

Data type change to two SYSCAT.REFERENCES columns, PK_COLNAMES and FK_COLNAMES, from VARCHAR(320) to VARCHAR(640).

Symptom:

Primary key and/or foreign key column names are truncated, are not correct, or are missing.

Explanation:

When column names greater than 18 bytes in length are used in a primary key or foreign key, the format under which the list of column names are stored in these two columns cannot remain the same. The 20-byte blank delimited column name(s) coming after the column whose length is greater than 18 will be shifted to the right the number of bytes that the column whose length is greater than 18 is over 18 bytes. As well, if the list of column names exceeds 640 bytes, the column will contain the empty string.

Resolution:

The view SYSCAT.KEYCOLUSE contains the list of columns that make up a primary, foreign, as well as a unique key and should be used instead of the columns in SYSCAT.REFERENCES. Alternatively, users can restrict the length of column names to 18 bytes or restrict the total length of the list of columns to 640 bytes.

SYSCAT.VIEWS Column TEXT in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

View text in the SYSCAT.VIEWS column TEXT will now not be split across multiple rows. The data type is changed from VARCHAR(3600) to CLOB(64K).

Symptom:

The complete view text is not given by the tool or in the application.

Explanation:

Tools or applications which were coded expecting no more than 3600 (or perhaps 3900) bytes to be returned from the TEXT column at one time are not handling the increased size of this field. The mechanism of retrieving multiple rows and reconstructing the view text using the SEQNO field is no longer necessary. The SEQNO value will now only ever be 1.

Resolution:

Change the tool or application to be able to handle values from the TEXT column which are greater than 3600 bytes. Alternatively, the view TEXT could be rewritten to fit within 3600 bytes.

SYSCAT.STATEMENTS Column TEXT in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

Statement text in the SYSCAT.STATEMENTS column TEXT will now not be split across multiple rows. The data type is changed from VARCHAR(3600) to CLOB(64K).

Symptom:

The complete statement text is not given by the tool or in the application.

Explanation:

Tools or applications which were coded expecting no more than 3600 (or perhaps 3900) bytes to be returned from the TEXT column at one time are not handling the increased size of this field. The mechanism of retrieving multiple rows and reconstructing the statement text using the SEQNO field is no longer necessary. The SEQNO value will now only ever be 1.

Resolution:

Change the tool or application to be able to handle values from the TEXT column which are greater than 3600 bytes. Alternatively, the statement TEXT could be rewritten to fit within 3600 bytes.

SYSCAT.INDEXES Column COLNAMES in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

The SYSCAT.INDEXES column COLNAMES data type is changed from VARCHAR(320) to VARCHAR(640).

Symptom:

Column names are missing in an index.

Explanation:

Tools or applications are coded to retrieve from a column with a data type of VARCHAR(320) and cannot handle the increased size of this field.

Resolution:

The view SYSCAT.INDEXCOLUSE contains the list of columns that make up an index and should be used instead of the column COLNAMES.
Alternatively, remove a column from the index or reduce the size of the name of a column so that the list of column names (with the leading + or −) will fit in 320 bytes.

SYSCAT.CHECKS Column TEXT in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

CHECKS Column TEXT data type is changed from CLOB(32K) to CLOB(64K).

Symptom:

Check constraint clause is incomplete.

Explanation:

Tools or applications are coded to retrieve from a column with a data type of CLOB(32K) and cannot handle the increased size of this field.

Resolution:

Change the tool or application to be able to handle values from the TEXT column which are longer than 32K bytes. Alternatively, rewrite the check constraint clause to use fewer characters such that it will fit in 32K bytes.

Column Data Type to BIGINT in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

Several system catalog view columns have had their data type changed from INTEGER to BIGINT.

Symptom:

Values are much smaller (or larger) than expected, especially statistical information.

Explanation:

Tools or applications are coded to retrieve from a column with a data type of INTEGER and cannot handle the increased size of this field.

Resolution:

Change the tool or application to be able to handle values which are greater than the maximum or minimum value which can be stored in an INTEGER field. Alternatively, change the underlying structure or SQL code which causes the value to be greater than what can be represented in an INTEGER field.

Column Mismatch in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

New columns are not inserted at the end of views in the SYSCAT view definition.

Symptom:

Re-preprocessing fails with several column mismatches or column data type mismatches.

Explanation:

New columns are introduced to the system catalog views and placed in a position that is useful in an ad-hoc query environment, specifically, shorter columns are placed before very long columns and the REMARKS column is always the last one.

Resolution:

Explicitly name the columns in the select list instead of coding “SELECT *”.

SYSCAT.COLUMNS and SYSCAT.ATTRIBUTES in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

SYSCAT.COLUMNS and SYSCAT.ATTRIBUTES now contain entries for inherited columns and attributes.

Symptom:

Queries of SYSCAT.COLUMNS to retrieve the columns of a typed table or view, and queries of SYSCAT.ATTRIBUTES to retrieve the attributes of a structured type, may return more rows in V6 than in V5.2, if the subject of the query is a subtable, subview, or subtype.

Explanation:

In Version 5.2, for a given table, view, or structured type, the COLUMNS and ATTRIBUTES catalogs contained entries only for columns and attributes that were introduced by that table, view, or type. Columns and attributes that were inherited from supertables or supertypes were not represented in the catalogs. However, in V6, the COLUMNS and ATTRIBUTES catalogs will contain entries for inherited columns and attributes.

Resolution:

Change the tool or application to recognize the new entries in the COLUMNS and ATTRIBUTES catalogs.

OBJCAT Views No Longer Supported in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

The recursive catalog views in the OBJCAT schema of Version 5.2 are no longer part of the shipped DB2 Universal Database product.

Symptom:

Queries written against the OBJCAT catalog views will no longer run successfully.

Resolution:

Most of the information formerly in the OBJCAT views has been incorporated into the regular SYSCAT catalog views. In most cases, you can obtain the information from the system catalog views. If you migrate from Version 5.2, and the OBJCAT catalog views exist, they should be dropped. This can be done by executing the CLP script called `objcatdp.db2` found under the `misc` subdirectory of the `sqllib` subdirectory.

If you wish, you could also create your own set of OBJCAT views that are equivalent to the catalog views supported in Version 5.2.

In version 5.2, the SQL Reference in Appendix E warned users that the OBJCAT catalog views were temporary and would not be supported in future releases.

Dependency Codes Changed in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

In the system catalog views, the hierarchic dependencies formerly denoted by code “H” are now denoted by code “O”.

Symptom:

Queries that search for hierarchic dependencies by code “H” in the catalog views will no longer work correctly.

Explanation:

Several system catalogs, including the system catalog views named PACKAGEDEP, TRIGDEP, and VIEWDEP have a column named BTYPE. In Version 5.2, the OBJCAT views denoted hierarchic dependencies with the code “H”. In Version 6, these dependencies are denoted with the code “O”.

Resolution:

These queries will need to be revised to search for code “O”.

SYSIBM Base Catalog Tables in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

The following are changes made to the SYSIBM base catalog tables, which customers may be using despite our encouragement for them to use the SYSCAT views. Customers who are not following our recommendation of coding to the SYSCAT views may experience incompatibilities due to the following changes:

1. Deleted fields (but still in the SYSCAT views):
   - SYSSTMT.SEQNO
   - SYSVIEWS.SEQNO

2. Renamed catalog table: SYSTRIGDEP named to SYSDEPENDENCIES. As well, the columns BCREATOR and DCREATOR were renamed to BSCHEMA and DSCHEMA respectively. The view SYSCAT.TRIGDEP did not change.
3. Deleted fields (were never in the SYSCAT views):
   - SYSATTRIBUTES.DEFAULT_VALUE
   - SYSATTRIBUTES.NULLS
   - SYSCOLUMNS.SERVERTYPE
   - SYSDATATYPES.REFREP_TYPENAME
   - SYSDATATYPES.REFREP_TYPESCHEMA
   - SYSDATATYPES.REFREP_LENGTH
   - SYSDATATYPES.REFREP_SCALE
   - SYSDATATYPES.REFREP_CODEPAGE
   - SYSINDEXES.TEXT

     (Was in the view, but reserved for future use only.)
   - SYSPLANDEP.PUBLICPRIV
   - SYSSECTION.SEQNO
   - SYSTABAUTH.UPDATE_BY_COLS
   - SYSTABAUTH.REF_BY_COLS
   - SYSTABLES.MINPDLENGTH
   - SYSTABLESPACES.READONLY
   - SYSTABLESPACES.REMOVABLEMEDIA
4. Data type changes:
   - SYSSECTION.SECTION from VARCHAR(3600) to CLOB(10M)
   - SYSPLANDEP.COLUSAGE from VARCHAR(3000) FOR BIT DATA to BLOB(5K)

Application Programming

VARCHAR Data Type in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

Maximum possible size of VARCHAR (VARGRAPHIC) data type has increased from 4 000 characters (2 000 double bytes characters), to 32 672 characters (16 336 double bytes characters) in Version 6.

Symptom:

An application that uses fixed length buffers of 4 000 bytes for a VARCHAR (VARGRAPHIC) data type has the potential for buffer overwrite or truncation, if it fetches a VARCHAR field which is longer than 4 000 bytes into a buffer that is too small. The CLI function - `SQLGetTypeInfo()` now returns the size of VARCHAR as 32 672. CLI applications that use this value in table DDLs may get errors due to sufficient page size table spaces not being available. See "User Table Data" on page 59 for more information on table space page size.

Resolution:

The application should be coded, in the recommended manner, of first describing the columns of the result set by using the DESCRIBE statement, and then using buffers whose size is based on the data type's length as returned from the DESCRIBE of the column.

Java Programming Positioned UPDATE and DELETE in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

When programming using Java in Version 6, positioned UPDATE and DELETE statements use the default authorization identifier of the person that bound the cursor package. This is different from Version 5.2 where the authorization identifier of the person running the package was used.

Symptom:

The package containing the positioned UPDATE and DELETE statements may not run because the authorization identifier of the person who bound the package does not have sufficient authority.

Resolution:

The authorization identifier of the person who binds the package must be granted sufficient authority to run the positioned UPDATE and DELETE statements found in the package. Grant the correct privileges and then re-bind the package.

Syntax Change in FOR UPDATE Clause in DB2 Universal Database Version 6

UNIX	OS/2	WIN	

Change:

In Version 5.2, in an SQLJ program, the FOR UPDATE clause can be used in a SELECT statement to identify the columns that can be updated in subsequent positioned UPDATE statements. The syntax is changed for Version 6.

Symptom:

You will receive the error message SQJ0204E if a SELECT statement contains a FOR UPDATE clause.

Resolution:

Remove the FOR UPDATE clause from the SELECT statement. Specify an updatable iterator through the iterator declaration clause. For example:

```
#sql public iterator DelByName implements sqlj.runtime.ForUpdate(String EmpNo)
   with updateColumns = (salary);
```

If you want to explicitly identify what columns are updateable, specify them through the `updateColumns` keyword, used in conjunction with the WITH clause.

Refer to *Application Development Guide* for more information on positioned iterator declarations.

Character Name Sizes in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

DB2 Universal Database Version 6 supports 128 byte table/view/alias names and 30 byte column names. The former support was for 18 byte names for each of these entities.

USER and CURRENT SCHEMA special registers were CHAR(8) and are now VARCHAR(128). The CURRENT EXPLAIN MODE special register was CHAR(8) and is now VARCHAR(254). The output for TYPE_SCHEMA and TABLE_SCHEMA built-in functions were CHAR(8) and are now VARCHAR(128).

Symptom:

If applications that were developed before Version 6 are run against a Version 6 database that does not utilize the longer limits, then the application behavior should not change at all. However, running these applications against a Version 6 database that does utilize longer names, could result in certain side effects, depending on how these applications were coded.

Here are some examples:

- Consider an existing application that FETCHes a table or column name (typically from a catalog view) into a host variable that was defined to be 18 bytes long. Since 18 bytes was the limit on the size of the table or column name until Version 6, this application may not bother to check the sqlwarn1 bit of the SQLCA, assuming that there should never be truncation. This application may proceed with subsequent logic, assuming that no truncation has occurred whereas truncation actually did occur.
- Consider an application that FETCHes a table or column name (typically from a catalog view) into an SQLDA where the size of the sqldata field was allocated based on the sqllen field from a DESCRIBE of the SELECT. This will result in the correct (untruncated) result being returned to the application even though the size of the table/column names may have increased. If other application logic relies on the assumption that column names or lengths are limited to 18 bytes, then the longer names that have been returned may introduce unintended behavior in the application. As a simple example, the display of a longer column name may be truncated at 18 bytes.
- Since the SQLCA token field (sqlerrmc) is limited to 70 bytes, this may affect existing applications that attempt to insert a row into a table. In the event of the error message SQL0204N, this application determines the name of the table from the SQLCA sqlerrmc field and then performs some operations based on that object name. Whereas with earlier versions of DB2, the table/schema identifier limit guaranteed that the table name was included in the SQLCA in its entirety (as in the case with the error message SQL0204N), this is not the case with Version 6.
- An application using a downlevel API (such as LIST HISTORY, or GET SNAPSHOT) will only get the first 18 bytes of a table name.
- Existing CLI and ODBC applications that use the schema functions (such as SQLTables(), or SQLColumns(), and others) will be affected if connecting to a server with support for greater than 18 byte names. Although there will be truncation warnings, the application may not check for this warning and may proceed with a truncated name.

Resolution:

The best way to resolve problems of this type is to re-code the application to handle longer table and column names. Otherwise, ensure that these applications are not run against Version 6 databases that use > 18 byte names.

PC/IXF Format Changes in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

DB2 Universal Database Version 6 supports 128 byte table/view/alias names and 30 byte column names. The former support was for 18 byte names for each of these entities.

Symptom:

A DB2 Universal Database Version 5 client is not able to import a PC/IXF that was exported by a DB2 Universal Database Version 6 export client. The error message SQL3059N will result.

Also, a PC/IXF file (an export from a DB2 Universal Database Version 6 client) cannot be loaded into a DB2 Universal Database Version 5 database. The error message SQL3059N will result.

Resolution:

Always be aware of the version level of the PC/IXF file when running.

SQLNAME in a Non-doubled SQLVAR in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

DB2 Universal Database Version 6 supports 30 byte column names. The former support was for 18 byte names. In Version 5, the documented behavior was that “0xFF” is placed in the 30th byte of an SQLNAME field for a non-doubled SQLVAR. Also in Version 5, for system-generated names and for user-specified column names specified in an “AS” clause, “0x00” is placed in the 30th byte.

In Version 6, the new behavior only returns “0xFF” in the 30th column if the name is system-generated.

Symptom:

Any applications that rely on the 30th-byte of the SQLNAME to determine whether it is a user-specified column name or a system-generated name may receive unexpected logic checks if the user-specifed column name is 30 characters long. This should be a rare occurrence.

Resolution:

These applications should be modified to only check for “0xFF” in the 30th byte of the SQLNAME field if the length of the SQLNAME is less than 30. In this case, the name is user-generated.

Obsolete DB2 CLI/ODBC Configuration Keywords in DB2 Universal Database Version 6

		WIN

Change:

When moving from version to version, 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 6, the TRANSLATEDLL and the TRANSLATEOPTION keywords became 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.

Event Monitor Output Stream Format in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

Event monitor output streams have no version control. As a result, adding support for the greater than 18 byte table names requires the move to an output stream format.

Symptom:

Applications that parse the event monitor output streams will no longer function as in previous releases.

Resolution:

There are two options:

1. Update the application to use the new data stream.
2. Set the registry variable DB2OLDEVMON=evmonname1,evmonname2,... where “evmonname” is the name of the event monitor you wish to have

write in the old data format. Note that any new fields in the event monitor will not be accessible in the old format.

SQL

Datalink Columns in DB2 Universal Database Version 6

UNIX		

Change:

Datalink values inserted in DB2 Universal Database Version 6 will require an extra four bytes of space in the column value descriptor.

Symptom:

When datalink columns created in Version 5.2 are updated, an additional four bytes are required on the data page to store the new column value. As a result, there may not be enough space in the data page to complete the update and it may have to be moved to a new page. This could cause the update to run out of space.

Resolution:

You will need to add more space on your system to allow for updates.

SYSFUN String Function Signatures in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

A number of string functions in the SYSFUN schema now have improved versions defined in the SYSIBM schema (built-in functions). The function names are LCASE, LTRIM, RTRIM, and UCASE.

Symptom:

When preparing statements or creating views, the returned data type from any of these functions may be different than running the same statement on versions previous to Version 6. This occurs because the built-in functions (under the SYSIBM schema) are usually resolved before functions in the SYSFUN schema.

Resolution:

No action is required. Usually, the behavior of the new built-in function is desired over the function as found in the SYSFUN schema. The previous

version behavior can be restored by switching the SQL path so that SYSFUN precedes SYSIBM, but performance of function resolution is degraded. Alternatively, the previous version function can be invoked by qualifying the function name with the schema name SYSFUN.

Migrated packages, views, summary tables, triggers, and constraints that reference these functions continue to use the version from the SYSFUN schema unless explicit action is taken such as explicitly binding the package or re-creating the view, summary table, trigger, or constraint.

SYSTABLE Column Change With New Integrity State in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

The "U" states in the CONST_CHECKED column of SYSCAT.TABLES changes differently when a SET INTEGRITY ... OFF statement is run.

Symptom:

Prior to Version 6, any "U" state in CONST_CHECKED column changes to an "N" when a SET INTEGRITY ... OFF statement is run. There is now another state, "W", to which the "U" state is changed.

Resolution:

No action is required.

The new "W" state in a CONST_CHECKED byte is used to indicate that the type of constraint was previously checked by the user and some data in the table may need to be checked for integrity.

Without this new state, the "U" state would be changed to the "N" state on a SET INTEGRITY ... OFF as it did in previous versions. From the "N" state alone, the database manager would not be sure if there exists any old data that has not yet been verified by the database manager. On a subsequent a SET INTEGRITY ... IMMEDIATE CHECKED INCREMENTAL statement, the database manager has to return an error. The error is returned because the database manager will not be able to guarantee data integrity if only the new change (if any) were checked.

With the new "W" state, on a subsequent a SET INTEGRITY ... IMMEDIATE CHECKED INCREMENTAL statement, the "W" state can be changed back to the "U" state if the INCREMENTAL option is specified to indicate that the user is still responsible for the data integrity of the table. If the

INCREMENTAL option is not specified, the database manager will pick full processing, change the “W” state to a “Y” state, and re-assume the responsibility of maintaining data integrity.

Database Security and Tuning

Creating Databases Using Clients in DB2 Universal Database Version 6

UNIX	OS/2	WIN	Extended

Change:

The method used by clients to create a database.

Symptom:

Using a down-level client (from a previous release than that of the server) to create a database will result in errors.

Resolution:

When using a client to create a database, only use clients at the same level as that of the server.

SELECT Privilege Required on Hierarchy in DB2 Universal Database Version 6

UNIX	OS/2	Windows 32-bit	Extended

Change:

Specification of the ONLY keyword with a table now requires that the user have SELECT privilege on all subtables of the specified typed table. Similarly, specification of the ONLY keyword with a view now requires that the user have SELECT privilege on all subviews of a specified typed table. Previous versions only required SELECT privilege on the specified table or view.

Symptom:

There are two possible symptoms:

1. An authorization error (SQLCODE -551, SQLSTATE 42501) occurs when rebinding a package containing an SQL statement that specified the ONLY keyword in a FROM clause, if the authorization ID under which the package was bound lacks SELECT privilege on the subtables of the named typed table (or view).
2. If the definition of a view or trigger contains the keyword ONLY used in a FROM clause, the view or trigger will continue to work normally.

However, the definition of the view or trigger can no longer be used to create a new view or trigger unless the creator holds SELECT privilege on all the subtables of the named table (or view).

Resolution:

The authorization ID that needs to rebind a package or to crate a new view or trigger should be granted SELECT privilege on all subtables (and subviews) of the table (or view) specified following the ONLY keyword.

Obsolete Profile Registry and Environment Variables in DB2 Universal Database Version 6

UNIX	OS/2	WIN	DB2 PE

Change:

The following profile registry values or environment variables are obsolete:

- DB2_VECTOR

Resolution:

There is no longer a need for these profile registry or environment variables.

Utilities and Tools

Current Explain Mode in DB2 Universal Database Version 6

UNIX	OS/2	WIN

Change:

The type of the “CURRENT EXPLAIN MODE” special register has changed from CHAR(8) to VARCHAR(254).

Symptom: If the application assumes that the type is still CHAR(8), then the value may be truncated from 254 to 8 bytes.

Resolution:

Redefine the type of all host variables which read the special register from CHAR(8) to VARCHAR(254).

This change is required to accommodate two new values for the “CURRENT EXPLAIN MODE” special register. These new values are “EVALUATE INDEXES” and “RECOMMEND INDEXES”.

The USING and SORT BUFFER Parameters in DB2 Universal Database Version 6

UNIX	OS/2	WIN	DB2 PE

Change:

As of Version 6, the USING and SORT BUFFER parameters of the LOAD command are no longer supported (the parameters are ignored).

Symptom:

If specified on the LOAD command, the use receives a warning message saying that USING and SORT BUFFER parameters are no longer supported and are ignored by the LOAD utility.

Resolution:

Ignore the warning message. See *Data Movement Utilities Guide and Reference* for additional information.

Connectivity and Coexistence

Replace RUMBA with PCOMM in DB2 Universal Database Version 6

		WIN

Change:

In Version 6, RUMBA is replaced by PCOMM on Windows NT, Windows 95, and Windows 98 only. RUMBA on Windows 3.1 will not be replaced.

Symptom:

None.

Resolution:

Windows 3.1 users will continue using RUMBA.

Configuration Parameters

Obsolete Database Configuration Parameters

UNIX	OS/2	WIN

Change:

The following database configuration parameters are obsolete with this version:

- DL_NUM_BACKUP (replaced by NUM_DB_BACKUP database configuration parameter)

Resolution:

Applications should be updated to not reference these parameters.

DB2 Universal Database Version 5 Incompatibilities

This section focuses on the DB2 Universal Database Version 5 incompatibilities.

System Catalog Views

System Catalog Views Column Changes

UNIX	OS/2	WIN

Change:

A variety of changes have been made to the system catalog views. 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.

Application Programming

External Table Functions in DB2 Universal Database Version 5

UNIX	OS/2	WIN	Extended

Change:

In Version 5.2, the user has been given explicit control over scratchpad duration.

For User Defined Functions (UDFs) which are table functions:

- FINAL CALL is now optional for the CREATE FUNCTION statement for table functions, as it is for scalar functions. NO FINAL CALL (the default) may thus be specified for table functions.
- When writing a table function which has a scratchpad, you can elect to have the scratchpad be initialized for each OPEN call to the function, by specifying NO FINAL CALL, in which case the table function is expected to release resources on each CLOSE call.

  Or, you can elect to have the scratchpad content be preserved across OPEN calls, by specifying FINAL CALL. If you do this, you need not release resources during the CLOSE call processing, because a FINAL call will be made to the table function at end-of-statement. The FINAL CALL is new with this change, as is the balancing FIRST call which takes place before the first OPEN call. These two new call types occur only on table functions which specify FINAL CALL.

- For Java table functions, there is a change to the execution model based on the new rules declared above. Refer to the *Application Development Guide* for details on the changes to table function processing.

Symptom:

The introduction of new call types may generate unexpected calls and may result in bad results being returned from the function depending on how the UDF is written to examine the call-type arguments.

Resolution:

Customers who have existing table functions should examine them knowing the changes introduced by the new call types and the new execution model. The following changes should be considered:

- If you depend on the new scratchpad rules implied by FINAL CALL then you may have to change your UDF to add the new call types, because with FINAL CALL specified, the new calls will be made to the UDF. Also, any acquired resources should be freed when the FINAL call is handled by the table function.
- If you wish to use the rules implied by NO FINAL CALL, you are advised to change and rerun your CREATE FUNCTION statement with NO FINAL CALL instead of FINAL CALL, so that your UDF will not get the new calls.
- If you have a Java table function, then you will have to change your function to conform to the new execution models. For either execution model (depending on whether you choose FINAL CALL or NO FINAL CALL), you will note that additional calls are made to the UDF method. This gives additional power to the UDF writer, but means that the writer must distinguish the calls. Use of the new `getCallType` method in the parent class included by DB2 provides this capability.

Refer to the *Application Development Guide* for detailed information on managing these changes.

NS, NW and NX Locks

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 behavior 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 uncommitted 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.

Symptom: The previous example showed what occurs with an uncommitted deletion. A similar situation could also arise when inserting new values.

For example, you could have a scenario where you are scanning a table using an index on a column and looking for a value greater than or equal to two, but less than or equal to six, while using an isolation level of RS. The existing values that qualify in this example are two, four, and six. Then another user inserts five. An NS lock is obtained on columns returning two, four, and six; and the NW lock attempt on the column containing six succeeds, so the insertion of five is not blocked by the scan.

In Version 2, an S lock would be obtained on columns with the values two, four, and six; and the attempt to get an X lock on the column returning six would wait. The insert of five would wait for the S lock on six to be released.

Resolution: In general, since more concurrency is supported in Version 5, applications built with a previous version of DB2 that were created with dependencies on some lock waiting may require modification.

DB2 Call Level Interface (DB2 CLI) Defaults

UNIX	OS/2	WIN

Change:

When moving from Version 2 to Version 5, 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

DB2 CLI SQLSTATEs

UNIX	OS/2	WIN

Change:

When moving from Version 2 to Version 5, 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/ODBC Configuration Keyword Defaults

UNIX	OS/2	WIN

Change:

When moving from Version 2 to Version 5, 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 in DB2 Universal Database Version 5

UNIX	OS/2	WIN

Change:

When moving from version to version, 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 became 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:

When moving from Version 2 to Version 5, 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:

When moving from Version 2 to Version 5, 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.

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 (-2 147 483 648) is a DECIMAL constant with a precision of 10 and a scale of 0 in Version 5.

Further, the smallest literal representation of a large INTEGER constant is -2 147 483 647 and not -2 147 483 648 (which is the limit for large INTEGER values). The INTEGER constant -2 147 483 648 is a BIGINT, not a DECIMAL (as it was before Version 5.2).

In general, if an integer constant is outside the range of a large integer and within the range of a BIGINT, then it is a BIGINT. If it is too big for a BIGINT, then it is a DECIMAL.

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.

Down-level Client and Distinct Types Sourced on BIGINT

UNIX	OS/2	WIN	DB2 PE

Change:

A distinct type based on BIGINT in a Version 5.2 server is reported in a DESCRIBE to a down-level client as a DECIMAL(19,0) instead of as a BIGINT which is not supported by the client. This data type cannot be implicitly cast

on assignment to the distinct type on which it is based. This is different than other situations where the client perceives a distinct type as a built-in data type and is able to assign host variables of the built-in type to columns of the associated distinct type.

Symptom:

An SQLCODE of -408 (SQLSTATE 42821) is returned when using a data type of DECIMAL(19,0) for the host variable (or parameter marker) assigned to the distinct type value that was described to the down-level client as DECIMAL(19,0).

Resolution:

The database should include a function that will cast a DECIMAL(19,0) to the distinct type. This can be defined as a sourced function based on the function that casts a BIGINT to the distinct type. The application (at the client) must then explicitly apply this function to the DECIMAL(19,0) host variable (or parameter marker) in the INSERT or UPDATE statement.

For example, if the distinct type sourced on BIGINT is called DT1, then updating the column C1 of type DT1 would require the following sourced function to be defined:

```
CREATE FUNCTION DT1(DECIMAL(19,0)) RETURNS DT1 SOURCE DT1(BININT);
```

And then the update statement in the application would be:

```
UPDATE table SET c1=DT1(:dechv1);
```

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 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.

Supported Level of JDBC

UNIX	OS/2	WIN

Change:

The supported level of the JDBC (Java Support) 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.

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 1.1.2 or Java 1.2 runtime environment at the client and server. At the server, set the `jdk11_path` configuration parameter.

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 moving from Parallel Edition Version 1. 2 to DB2 Universal Database Version 5, and 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 4 005 bytes. If temporary tables are close to the 4 005 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 affect 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:

In Version 5, the authorization required for creating or dropping a nodegroup changed from SYSADM or DBADM to SYSADM or SYSCTRL. 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:

Beginning in Version 5, 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.

Database Security and Tuning

Creating Databases Using Clients in DB2 Universal Database Version 5

UNIX	OS/2	WIN	Extended

Change:

The method used by clients to create a database.

Symptom:

Using a down-level client (from a previous release than that of the server) to create a database will result in errors.

Resolution:

When using a client to create a database, only use clients at the same level as that of the server.

Utilities and Tools

LOAD TERMINATE

			DB2 PE

Change:

The LOAD TERMINATE command has a different function in DB2 UDB than it did in DB2 Parallel Edition Version 1.x. In DB2 Parallel Edition, you could use LOAD TERMINATE if an error occurred during the load operation to ensure that the table data was consistent. In DB2 Universal Database Version 5 however, if you use LOAD TERMINATE, the table space is moved into the recovery pending state. (When the table space is in the recovery pending state, you must either restore the table space or drop it.)

Symptom:

Instead of being placed in a consistent state, the table space is placed in a recovery pending state.

Resolution:

Instead of using LOAD TERMINATE to clean up after a failed load operation, you should use LOAD RESTART or LOAD REPLACE. You also have the option of dropping and re-creating the table space.

REORG - Alternate Path Option

UNIX	OS/2	

Change:

When moving from Version 2 to Version 5, 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

There are no incompatibilities for this area.

Configuration Parameters

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:

Re-code 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.

BUFFPAGE and Multiple Buffer Pools

UNIX	OS/2	WIN	DB2 PE

Change:

Before Version 5, 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 and later, each database can have multiple buffer pools. You can create additional buffer pools of possibly different page sizes 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.

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.

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 $

Note: You may be required to perform password maintenance tasks. Since such tasks are required at the server, and many of the users are not able or comfortable working with the server environment, carrying these tasks can pose a significant challenge. DB2 UDB provides a way to update and verify passwords without having to be at the server. For example, DB2 for OS/390 Version 5 supports this method of changing a user's password. If an error message SQL1404N "Password expired" is received, then to change the password use the CONNECT statement as follows:

```
CONNECT TO <database> USER <userid> USING
   <password> NEW <new_password>
   VERIFY <new_password>
```

The "Password change" dialogue of the DB2 Client Configuration Assistant (CCA) may also be used to make a change to the password. See the *SQL Reference* and the CCA online help for further information on these methods to change the password.

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: There are exceptions:
  - Schemas only allow 1 to 8 characters
  - Columns allow 1 to 30 characters
  - Tables, views, and aliases allow 1 to 128 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 H. National Language Support (NLS)" on page 745.

Federated Database Object Names

Federated database objects include:

- Index specifications
- Nicknames
- Servers
- Wrappers
- Function mappings
- Type mappings
- User mappings

Limits apply when naming federated database objects. A complete list of object names and associated identifier limits and requirements are located in the *SQL Reference*. In summary, object names:

- Have limits. Mapping, index specification, server, wrapper, and nickname names cannot exceed 128 bytes.
- Must begin with one of the following:
  - A through Z (names without quotes are converted to uppercase)
  - A valid accented letter (such as ö)
  - A multibyte character, except multibyte spaces (for multibyte environments)
- Must follow internal naming conventions. Non-leading characters can include:

- A through Z
- 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.

Options (server, nickname) and option settings are limited to 255 bytes.

How Case-Sensitive Values Are Preserved in a Federated System

In distributed requests, you sometimes need to specify identifiers and passwords that are case-sensitive at the data source. To ensure that their case is correct when they're passed to the data source, follow these guidelines:

- Specify them in the required case and enclose them in double quotes.
- If you're specifying a user ID, set the fold_id server option to 'n' ("No, don't change case") for the data source. If you're specifying a password, set the fold_pw server option to 'n' for the data source.

  There is an alternative for user IDs and passwords. If a data source requires a user ID to be in lowercase, you can specify it in any case and set the fold_id server option to 'l' ("Send this ID to the data source in lowercase"). If the data source requires the ID to be in uppercase, you can specify it in any case and set fold_id to 'u' ("Send this ID to the data source in upper case"). In the same way, if a data source requires a password to be in lowercase or uppercase, you can meet this requirement by setting the fold_pw server option to 'l' or 'u'.

  For more information about server options, see "Using Server Options to Help Define Data Sources and Facilitate Authentication Processing" on page 192.
- If you enclose a case-sensitive identifier or password in double quotes at an operating system's command prompt, you need to ensure that the system parses the double quotes correctly. To do this:
  - On a UNIX operating system, enclose the statement in single quotes.
  - On a Windows NT operating system, precede each quote with a backward slash.

For example, many delimited identifiers in DB2 family data sources are case-sensitive. Suppose you want to create a nickname, NICK1, for a DB2 for CS view, "my_schema"."wkly_sal", that resides in a data source called NORBASE. If you're entering the SQL for creating the nickname from a UNIX command prompt, you would type:

```
db2
'create nickname nick1 for norbase."my_schema"."wkly_sal"'
```

From an NT command prompt, you would type:

```
db2 create nickname nick1 for norbase.\"my_schema\".\"wkly_sal\"
```

If you enter the SQL from the DB2 interactive mode command prompt, or if you specify it in an application program, you don't need the single quotes or the slashes. For example, from the DB2 command prompt on either a UNIX or NT system, you would type:

```
create nickname nick1 for norbase."my_schema"."wkly_sal"
```

Appendix E. 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 Registry Variables
- CATALOG and ATTACH Commands, and the CONNECT Statement
- 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" on page 700
- "Database Locator Objects" on page 701
- "Routing Information Objects" on page 703

Each object contains attributes. Refer to "Attributes of Each Object Class" on page 704 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 721 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 713 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 721.

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, Windows NT, or Windows 95).

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 721.

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, Windows NT, or Windows 95).

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 73 on page 715.

```
   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 721.

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, Windows NT, or Windows 95).

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 shows the attributes for each object class and Table 41 on page 705 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

Table 40. Object Attribute Classes (continued)

Object Class	Object ID (OID)	Required Attributes	Optional Attributes
(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, Windows NT, or Windows 95 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, Windows NT, or Windows 95 only) The tokens for NetBIOS are:
  1. netbios
  2. Node name of the server

  For example: `netbios;SVR_NNME` where the client adapter number is found in either the registry value *db2clientadpt* or the database manager configuration parameter *dft_client_adpt*.
- (Windows NT or Windows 95 only) The tokens for NPIPE are:
  1. NPIPE
  2. Computer name of the server
  3. Instance name of the server

  For example: `npipe;computername;instance`

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 710 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 clients accessing database servers. For more information, see "Authentication" on page 282.

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 on page 710 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 Registry Variables

The following configuration parameters are used with DCE directories. An example of the values is shown. Refer to "Distributed Services" within the chapter "Configuring DB2" in *Administration Guide, Performance* 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 DCE directory services 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 DCE 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)
  TCPIP,APPC,IPXSPX,NETBIOS,NPIPE    (on Windows NT or Windows 95 platforms)
  ```

- *dft_client_adpt* is an optional DCE parameter that specifies the default client adapter number for the NetBIOS protocol on OS/2, Windows NT, or Windows 95. 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 defaults to zero (0).

For the following parameters, registry variables can override the parameter values.

Configuration Parameter	**Registry Variable**
dir_path_name	DB2DIRPATHNAME
route_obj_name	DB2ROUTE
dft_client_comm	DB2CLIENTCOMM
dft_client_adpt	DB2CLIENTADPT

The rules for setting these registry 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:

```
db2set DB2CLIENTCOMM=TCPIP,APPC
```

CATALOG and ATTACH Commands, and the CONNECT Statement

DCE information needs to be specified in the following commands:

- CATALOG GLOBAL DATABASE Command
- CONNECT Statement
- 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 Statement

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 statement 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 registry value.)

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 73 on page 715 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.)

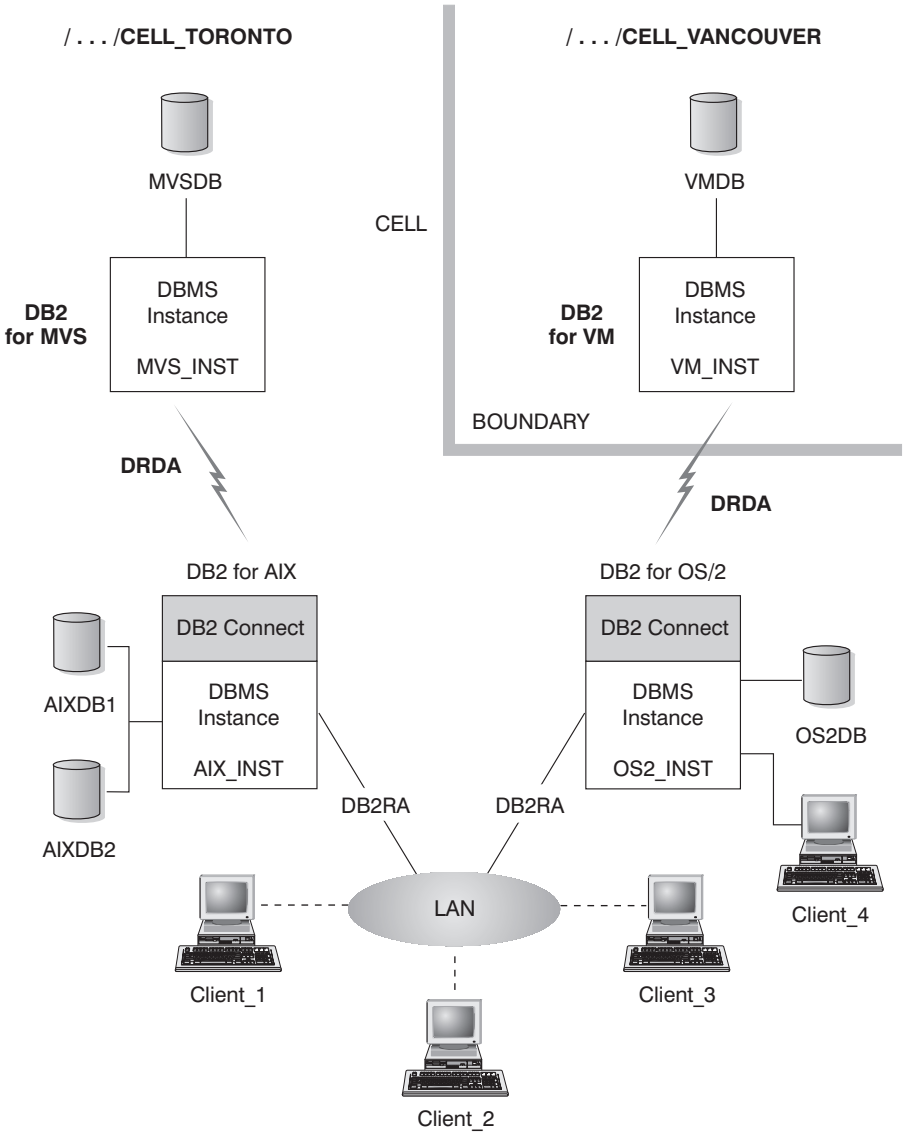


Figure 73. Configuration of A Network Database

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 registry value).
   - 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 registry value).

   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 registry value) 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 registry value).
- 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 registry value).
- Specify the DCE name of the default routing information object in the configuration parameter *route_obj_name* (or the DB2ROUTE registry value).

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 registry value. 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 registry value).
- Specify the DCE name of the default routing information object in the configuration parameter *route_obj_name* (or the DB2ROUTE registry value).

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 registry value. 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 Statement

ATTACH Command

Figure 74 on page 719 shows how the directories are searched when a client attaches to a DBMS instance called ABC_INST.

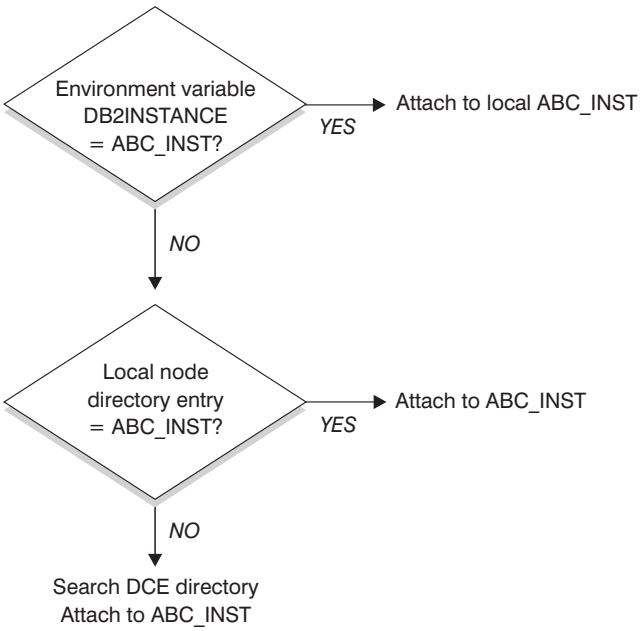


Figure 74. How Directories are Used to Attach a Database

CONNECT Statement

Figure 75 on page 720 shows how the directories are searched when a client connects to a database called DBTEST.

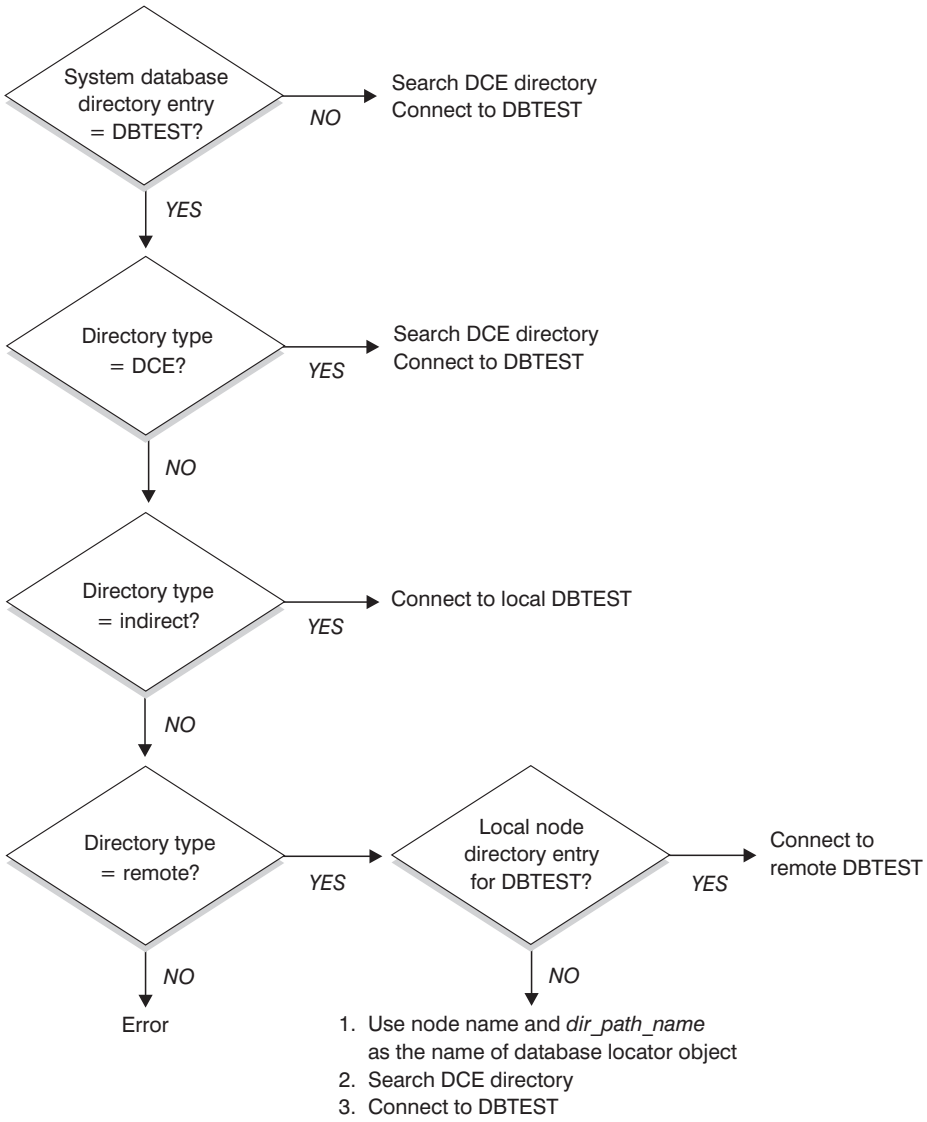


Figure 75. How Directories are Used to Connect a Database

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 713 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 registry values set by the client. Refer to "Configuration Parameters and Registry Variables" on page 711 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 699 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 registry values. Registry values 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 registry value 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 registry value. Here are **some** UNIX examples:

    ```
    db2set DB2CLIENTCOMM=tcpip
    db2set DB2CLIENTCOMM=appc
    db2set DB2CLIENTCOMM=tcpip,appc
    db2set DB2CLIENTCOMM=appc,tcpip
    ```

    Some OS/2 examples are:

    ```
    db2set DB2CLIENTCOMM=ipxspx
    db2set DB2CLIENTCOMM=netbios
    db2set DB2CLIENTCOMM=tcpip,ipxspx,netbios
    db2set DB2CLIENTCOMM=netbios,tcpip,ipxspx,appc
    ```

    Some Windows NT and Windows 95 examples are:

```
db2set DB2CLIENTCOMM=npipe
db2set DB2CLIENTCOMM=netbios
db2set DB2CLIENTCOMM=tcpip,ipxspx,netbios
db2set DB2CLIENTCOMM=netbios,tcpip,ipxspx,appc,npipe
```

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 registry value, then explicitly catalog the database with the CATALOG GLOBAL DATABASE command. Refer to "CATALOG GLOBAL DATABASE Command" on page 713 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 Clients for all UNIX, OS/2, Windows NT, and Windows 95 platforms.
- A client cannot use DCE Directory Services to connect to a DB2 for OS/2 Version 1 server.
- Only Windows NT or Windows 95 clients can use any or all of the TCP/IP, APPC, NetBIOS, IPX/SPX, or NPIPE protocols. Only OS/2 clients can use any or all of the TCP/IP, APPC, NetBIOS, and IPX/SPX protocols. All supported UNIX clients 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.
- When using Windows NT, Windows 95, or Windows 98 clients, DB2DCE.DLL will be used. This file is found in the *bin* subdirectory of the *sqllib* subdirectory. If the DCE provider is Gradient**, by default the file DB2DCE.GRD is equivalent to DB2DCE.DLL. If the DCE provider is IBM, the file DB2DCE.IBM must be copied to DB2DCE.DLL.

Appendix F. 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.

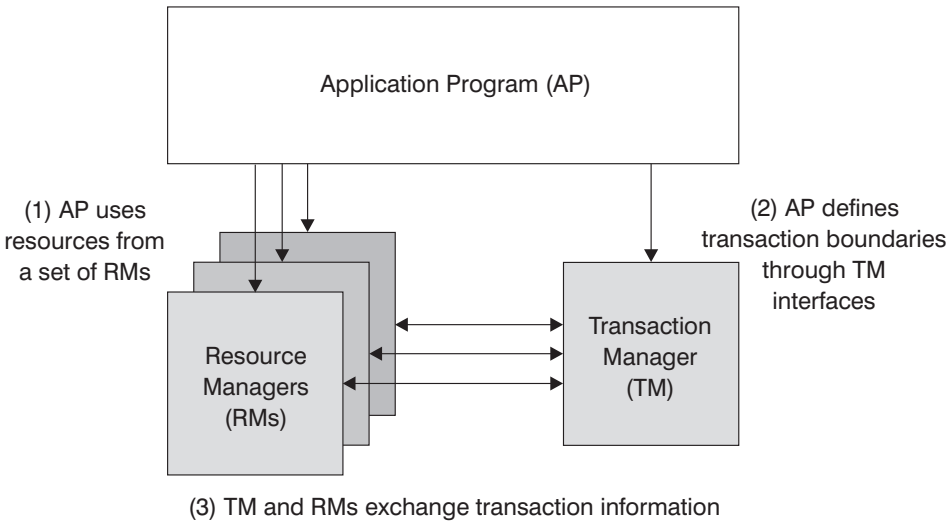


Figure 76. X/Open Distributed Transaction Processing (DTP) Model

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).

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 494 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 469.)

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" on page 496.

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 *Application Development 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.

Appendix G. 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 394 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 400 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 413).

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 413).

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 Seagate** Software Inc., to store and retrieve data on an IBM external tape device.

  Note: Only Version 2.2 of the Sytos Premium product is currently supported. You require the OS/2 FixPack 26 to use this product.

  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 four 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.
- **db2uxt2.cxbsa**

  This program uses the Legato NetWorker** Version 4.2.5 utility from Legato** Systems, Incorporated to archive and retrieve database log files. This program is only applicable to AIX.

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> -SP<startpage>
-LS<logsize>
```

os Platform on which the instance is running: AIX, NT, SUN, HP, SNI, SCO, 95, 98, 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.

startpage Log extent starts at this number of offset 4 KB pages of the device.

logsize The size of this log extent in 4 KB pages.

Notes:

1. Windows NT only supports user exits for archiving logs.
2. The **-LS** and **-SP** parameters are only used if a raw device is used for logging. If you are using an existing user exit program that uses files for logging, you do not have to change it.

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.

  Notes:

  1. Table space roll-forward recovery does not support the retrieval of log files using user exits.
  2. On Windows NT, you cannot use a REXX user exit to archive logs.
- 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.
- DB2 opens a file in read mode when it starts a user exit to archive a log file. Therefore, the user exit should not be able to delete the file while the

file is still active. DB2 closes the file when it becomes inactive. If the user exit finishes when the file is inactive, the log file can be deleted but there is a performance cost for doing so.

- When a log file has been archived, and it is inactive, DB2 does not delete the file but renames it as the next log file when such a file is needed. This results in a performance gain since when creating a new log file (instead of renaming the file), all pages must be written out to guarantee the disk space. It is better to re-use than to free up and then re-acquire the necessary pages on disk.
- DB2 will NOT invoke the user exit to retrieve the log file in either crash recovery nor rollback.
- 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 offline 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 429.)
- 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 735.)
- 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 44 on page 743 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 44. 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. The request for an archive involving the same log file is retried in five minutes.
3. User exit program requests are suspended for five minutes. During this time, all requests are ignored including the log file request that caused the return code.

   Following the five minute suspension in processing requests, the next request is processed. If no error occurs with the processing of this request, then processing of new user exit program requests continues. If a return code of greater than 8 is generated during the retry, requests are suspended for an additional five minutes. The five minute suspensions continue until the problem is corrected or the database is stopped and restarted.

   Once all applications disconnect from the database and then the database is reopened, the request involving the log file that originated the problem will be repeated. If successfully processed, any additional requests generated during the time of suspension are processed.

   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 H. 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 registry variable is set, its value is taken as the application code page. However, it is not necessary to set the DB2CODEPAGE registry variable because DB2 will determine the appropriate code page value from the operating system. Setting the DB2CODEPAGE registry variable to incorrect values may cause unpredictable results.

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 setting, which includes 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 32-bit operating systems
For all Windows 32-bit operating systems, if the DB2CODEPAGE environment variable is not set, the 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 Table 45 on page 747.

Deriving Locales in Application Programs

Locales are implemented one way in Windows and another in UNIX-based systems. In UNIX-based 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.

In Windows, the cultural preferences can be set through Regional Settings of the Control Panel. However, there is no environmental locale like the one on UNIX systems.

When your program is started, it gets a default C locale. It does **not** get a copy of the environment locale. If you set the program locale to any locale other than "C", DB2 Universal Database uses your current program locale to determine the code page and territory settings for your application environment. Otherwise, these values are obtained from the operating system environment. You should note that `setlocale()` is not thread-safe, and if you issue `setlocale()` from within your application, the new locale is set for the entire process.

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 assume the default of the US English environment, and code page 819 (ISO 8859-1).
- If LC_CTYPE's value is no longer 'C', its new value will be used to map to a corresponding code page. For information on the default locale for a

particular platform, see Table 45. Additional information may be found in the *Application Building Guide.*

Country Code and Code Page Support

Table 45 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. **Group** shows whether a code page is single-byte ("S") or multi-byte ("D"). The "-n" is a number used to create a letter-number combination. Matching combinations show where connection and conversion is allowed by DB2. For example, all "S-1" groups can work together.
3. **Code Set** shows the code set associated with the supported language. The code set is mapped to the DB2 Code Page.
4. **Tr.** shows the two letter territory identifier.
5. **Country Code** shows the country code that is used by the database manager internally for providing country-specific support.
6. **Locale** shows the locale values supported by the database manager.
7. **OS** shows the operating system that supports the languages and code sets.
8. **Country Name** shows the name of the country or countries.

Table 45. Supported Languages and Code Sets

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
437	S-1	IBM-437	AL	355	-	OS2	Albania
850	S-1	IBM-850	AL	355	-	OS2	Albania
819	S-1	ISO8859-1	AL	355	sq_AL	AIX	Albania
850	S-1	IBM-850	AL	355	Sq_AL	AIX	Albania
819	S-1	iso88591	AL	355	-	HP	Albania
1051	S-1	roman8	AL	355	-	HP	Albania
819	S-1	ISO8859-1	AL	355	-	Sun	Albania
1252	S-1	1252	AL	355	-	WIN	Albania
1275	S-1	1275	AL	355	-	Mac	Albania
37	S-1	IBM-037	AL	355	-	HOST	Albania
1140	S-1	IBM-1140	AL	355	-	HOST	Albania

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
864	S-6	IBM-864	AA	785	-	OS2	Arabic Countries
1046	S-6	IBM-1046	AA	785	Ar_AA	AIX	Arabic Countries
1089	S-6	ISO8859-6	AA	785	ar_AA	AIX	Arabic Countries
1089	S-6	iso88596	AA	785	ar_SA.iso88596	HP	Arabic Countries
1256	S-6	1256	AA	785	-	WIN	Arabic Countries
420	S-6	IBM-420	AA	785	-	HOST	Arabic Countries
437	S-1	IBM-437	AU	61	-	OS2	Australia
850	S-1	IBM-850	AU	61	-	OS2	Australia
819	S-1	ISO8859-1	AU	61	en_AU	AIX	Australia
850	S-1	IBM-850	AU	61	En_AU	AIX	Australia
819	S-1	iso88591	AU	61	-	HP	Australia
1051	S-1	roman8	AU	61	-	HP	Australia
819	S-1	ISO8859-1	AU	61	en_AU	Sun	Australia
819	S-1	ISO8859-1	AU	61	en_AU	SCO	Australia
1252	S-1	1252	AU	61	-	WIN	Australia
1275	S-1	1275	AU	61	-	Mac	Australia
37	S-1	IBM-037	AU	61	-	HOST	Australia
1140	S-1	IBM-1140	AU	61	-	HOST	Australia
437	S-1	IBM-437	AT	43	-	OS2	Austria
850	S-1	IBM-850	AT	43	-	OS2	Austria
819	S-1	ISO8859-1	AT	43	ge_AT	AIX	Austria
850	S-1	IBM-850	AT	43	Ge_AT	AIX	Austria
819	S-1	iso88591	AT	43	-	HP	Austria
1051	S-1	roman8	AT	43	-	HP	Austria
819	S-1	ISO8859-1	AT	43	de_AT	SCO	Austria
819	S-1	ISO8859-1	AT	43	de_AT	Sun	Austria
1252	S-1	1252	AT	43	-	WIN	Austria
1275	S-1	1275	AT	43	-	Mac	Austria
37	S-1	IBM-037	AT	43	-	HOST	Austria
1140	S-1	IBM-1140	AT	43	-	HOST	Austria
915	S-11	ISO8859-5	BY	375	-	OS2	Belarus
915	S-11	ISO8859-5	BY	375	be_BY	AIX	Belarus
1131	S-11	IBM-1131	BY	375	-	OS2	Belarus
1251	S-11	1251	BY	375	-	WIN	Belarus
1283	S-11	1283	BY	375	-	Mac	Belarus
1025	S-11	IBM-1025	BY	375	-	HOST	Belarus

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
437	S-1	IBM-437	BE	32	-	OS2	Belgium
850	S-1	IBM-850	BE	32	-	OS2	Belgium
819	S-1	ISO8859-1	BE	32	nl_BE fr_BE	AIX	Belgium
850	S-1	IBM-850	BE	32	Nl_BE Fr_BE	AIX	Belgium
819	S-1	iso88591	BE	32	-	HP	Belgium
819	S-1	ISO8859-1	BE	32	fr_BE	SCO	Belgium
819	S-1	ISO8859-1	BE	32	nl_BE	SCO	Belgium
819	S-1	ISO8859-1	BE	32	nl_BE fr_BE	Sun	Belgium
1252	S-1	1252	BE	32	-	WIN	Belgium
1275	S-1	1275	BE	32	-	Mac	Belgium
500	S-1	IBM-500	BE	32	-	HOST	Belgium
1148	S-1	IBM-1148	BE	32	-	HOST	Belgium
855	S-5	IBM-855	BG	359	-	OS2	Bulgaria
915	S-5	ISO8859-5	BG	359	-	OS2	Bulgaria
915	S-5	ISO8859-5	BG	359	bg_BG	AIX	Bulgaria
915	S-5	iso88595	BG	359	bg_BG.iso88595	HP	Bulgaria
1251	S-5	1251	BG	359	-	WIN	Bulgaria
1283	S-5	1283	BG	359	-	Mac	Bulgaria
1025	S-5	IBM-1025	BG	359	-	HOST	Bulgaria
850	S-1	IBM-850	BR	55	-	OS2	Brazil
850	S-1	IBM-850	BR	55	-	AIX	Brazil
819	S-1	ISO8859-1	BR	55	pt_BR	AIX	Brazil
819	S-1	ISO8859-1	BR	55	-	HP	Brazil
819	S-1	ISO8859-1	BR	55	pt_BR	SCO	Brazil
819	S-1	ISO8859-1	BR	55	pt_BR	Sun	Brazil
1252	S-1	1252	BR	55	-	WIN	Brazil
37	S-1	IBM-037	BR	55	-	HOST	Brazil
1140	S-1	IBM-1140	BR	55	-	HOST	Brazil
850	S-1	IBM-850	CA	1	-	OS2	Canada
850	S-1	IBM-850	CA	1	En_CA	AIX	Canada
819	S-1	ISO8859-1	CA	1	en_CA	AIX	Canada
819	S-1	iso88591	CA	1	fr_CA.iso88591	HP	Canada
1051	S-1	roman8	CA	1	fr_CA.roman8	HP	Canada
819	S-1	ISO8859-1	CA	1	en_CA	SCO	Canada
819	S-1	ISO8859-1	CA	1	fr_CA	SCO	Canada
819	S-1	ISO8859-1	CA	1	en_CA	Sun	Canada
819	S-1	ISO8859-1	CA	1	en_CA	Sun	Canada
1252	S-1	1252	CA	1	-	WIN	Canada
1275	S-1	1275	CA	1	-	Mac	Canada
37	S-1	IBM-037	CA	1	-	HOST	Canada
1140	S-1	IBM-1140	CA	1	-	HOST	Canada
863	S-1	IBM-863	CA	2	-	OS2	Canada (French)

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
1381	D-4	IBM-1381	CN	86	-	OS2	China (PRC)
1386	D-4	GBK	CN	86	-	OS2	China (PRC)
1383	D-4	IBM-eucCN	CN	86	zh_CN	AIX	China (PRC)
1386	D-4	GBK	CN	86	Zh_CN.GBK	AIX	China (PRC)
1383	D-4	hp15CN	CN	86	zh_CN.hp15CN	HP	China (PRC)
1383	D-4	eucCN	CN	86	zh_CN	SCO	China (PRC)
1383	D-4	eucCN	CN	86	zh_CN.eucCN	SCO	China (PRC)
1383	D-4	gb2312	CN	86	zh chinese	Sun	China (PRC)
1381	D-4	IBM-1381	CN	86	-	WIN	China (PRC)
1386	D-4	GBK	CN	86	-	WIN	China (PRC)
935	D-4	IBM-935	CN	86	-	HOST	China (PRC)
1388	D-4	IBM-1388	CN	86	-	HOST	China (PRC)
852	S-2	IBM-852	HR	385	-	OS2	Croatia
912	S-2	ISO8859-2	HR	385	hr_HR	AIX	Croatia
912	S-2	iso88592	HR	385	hr_HR.iso88592	HP	Croatia
912	S-2	ISO8859-2	HR	385	hr_HR.ISO8859-2	SCO	Croatia
1250	S-2	1250	HR	385	-	WIN	Croatia
1282	S-2	1282	HR	385	-	Mac	Croatia
870	S-2	IBM-870	HR	385	-	HOST	Croatia
852	S-2	IBM-852	CZ	42	-	OS2	Czech Republic
912	S-2	ISO8859-2	CZ	42	cs_CZ	AIX	Czech Republic
912	S-2	iso88592	CZ	42	cs_CZ.iso88592	HP	Czech Republic
912	S-2	ISO8859-2	CZ	42	cs_CZ.ISO8859-2	SCO	Czech Republic
1250	S-2	1250	CZ	42	-	WIN	Czech Republic
1282	S-2	1282	CZ	42	-	Mac	Czech Republic
870	S-2	IBM-870	CZ	42	-	HOST	Czech Republic
850	S-1	IBM-850	DK	45	-	OS2	Denmark
819	S-1	ISO8859-1	DK	45	da_DK	AIX	Denmark
850	S-1	IBM-850	DK	45	Da_DK	AIX	Denmark
819	S-1	iso88591	DK	45	da_DK.iso88591	HP	Denmark
1051	S-1	roman8	DK	45	da_DK.roman8	HP	Denmark
819	S-1	ISO8859-1	DK	45	da	SCO	Denmark
819	S-1	ISO8859-1	DK	45	da_DA	SCO	Denmark
819	S-1	ISO8859-1	DK	45	da_DK	SCO	Denmark
819	S-1	ISO8859-1	DK	45	da	Sun	Denmark
819	S-1	ISO8859-1	DK	45	da	Sun	Denmark
1252	S-1	1252	DK	45	-	WIN	Denmark
1275	S-1	1275	DK	45	-	Mac	Denmark
277	S-1	IBM-277	DK	45	-	HOST	Denmark
1142	S-1	IBM-1142	DK	45	-	HOST	Denmark

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
922	S-10	IBM-922	EE	372	-	OS2	Estonia
922	S-10	IBM-922	EE	372	Et_EE	AIX	Estonia
922	S-10	IBM-922	EE	372	-	WIN	Estonia
1122	S-10	IBM-1122	EE	372	-	HOST	Estonia
437	S-1	IBM-437	FI	358	-	OS2	Finland
850	S-1	IBM-850	FI	358	-	OS2	Finland
819	S-1	ISO8859-1	FI	358	fi_FI	AIX	Finland
850	S-1	IBM-850	FI	358	Fi_FI	AIX	Finland
819	S-1	iso88591	FI	358	fi_FI.iso88591	HP	Finland
819	S-1	ISO8859-1	FI	358	fi	SCO	Finland
819	S-1	ISO8859-1	FI	358	fi_FI	SCO	Finland
819	S-1	ISO8859-1	FI	358	sv_FI	SCO	Finland
819	S-1	ISO8859-1	FI	358	-	Sun	Finland
1051	S-1	roman8	FI	358	-	HP	Finland
1252	S-1	1252	FI	358	-	WIN	Finland
1275	S-1	1275	FI	358	-	Mac	Finland
278	S-1	IBM-278	FI	358	-	HOST	Finland
1143	S-1	IBM-1143	FI	358	-	HOST	Finland
855	S-5	IBM-855	MK	389	-	OS2	FYR Macedonia
915	S-5	ISO8859-5	MK	389	-	OS2	FYR Macedonia
915	S-5	ISO8859-5	MK	389	mk_MK	AIX	FYR Macedonia
915	S-5	iso88595	MK	389	-	HP	FYR Macedonia
1251	S-5	1251	MK	389	-	WIN	FYR Macedonia
1283	S-5	1283	MK	389	-	Mac	FYR Macedonia
1025	S-5	IBM-1025	MK	389	-	HOST	FYR Macedonia
437	S-1	IBM-437	FR	33	-	OS2	France
850	S-1	IBM-850	FR	33	-	OS2	France
819	S-1	ISO8859-1	FR	33	fr_FR	AIX	France
850	S-1	IBM-850	FR	33	Fr_FR	AIX	France
819	S-1	iso88591	FR	33	fr_FR.iso88591	HP	France
1051	S-1	roman8	FR	33	fr_FR.roman8	HP	France
819	S-1	ISO8859-1	FR	33	fr	Sun	France
819	S-1	ISO8859-1	FR	33	fr	SCO	France
819	S-1	ISO8859-1	FR	33	fr_FR	SCO	France
1252	S-1	1252	FR	33	-	WIN	France
1275	S-1	1275	FR	33	-	Mac	France
297	S-1	IBM-297	FR	33	-	HOST	France
1147	S-1	IBM-1147	FR	33	-	HOST	France

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
437	S-1	IBM-437	DE	49	-	OS2	Germany
850	S-1	IBM-850	DE	49	-	OS2	Germany
819	S-1	ISO8859-1	DE	49	de_DE	AIX	Germany
850	S-1	IBM-850	DE	49	De_DE	AIX	Germany
819	S-1	iso88591	DE	49	de_DE.iso88591	HP	Germany
1051	S-1	roman8	DE	49	de_DE.roman8	HP	Germany
819	S-1	ISO8859-1	DE	49	de	SCO	Germany
819	S-1	ISO8859-1	DE	49	de_DE	SCO	Germany
819	S-1	ISO8859-1	DE	49	de	Sun	Germany
1252	S-1	1252	DE	49	-	WIN	Germany
1275	S-1	1275	DE	49	-	Mac	Germany
273	S-1	IBM-273	DE	49	-	HOST	Germany
1141	S-1	IBM-1141	DE	49	-	HOST	Germany
819	S-1	ISO8859-1	DE	49	De_DE.88591	SINIX	Germany
819	S-1	ISO8859-1	DE	49	De_DE.6937	SINIX	Germany
NOTE: DB2 supports as ISO 8859-1; it should be ISO 6937							
813	S-7	ISO8859-7	GR	30	-	OS2	Greece
869	S-7	IBM-869	GR	30	-	OS2	Greece
813	S-7	ISO8859-7	GR	30	el_GR	AIX	Greece
813	S-7	iso88597	GR	30	el_GR.iso88597	HP	Greece
813	S-7	ISO8859-7	GR	30	el_GR.ISO8859-7	SCO	Greece
737	S-7	737	GR	30	-	WIN	Greece
1253	S-7	1253	GR	30	-	WIN	Greece
1280	S-7	1280	GR	30	-	Mac	Greece
423	S-7	IBM-423	GR	30	-	HOST	Greece
875	S-7	IBM-875	GR	30	-	HOST	Greece
852	S-2	IBM-852	HU	36	-	OS2	Hungary
912	S-2	ISO8859-2	HU	36	hu_HU	AIX	Hungary
912	S-2	iso88592	HU	36	hu_HU.iso88592	HP	Hungary
912	S-2	ISO8859-2	HU	36	hu_HU.ISO8859-2	SCO	Hungary
1250	S-2	1250	HU	36	-	WIN	Hungary
1282	S-2	1282	HU	36	-	Mac	Hungary
870	S-2	IBM-870	HU	36	-	HOST	Hungary
850	S-1	IBM-850	IS	354	-	OS2	Iceland
819	S-1	ISO8859-1	IS	354	is_IS	AIX	Iceland
850	S-1	IBM-850	IS	354	Is_IS	AIX	Iceland
819	S-1	iso88591	IS	354	is_IS.iso88591	HP	Iceland
1051	S-1	roman8	IS	354	is_IS.roman8	HP	Iceland
819	S-1	ISO8859-1	IS	354	is	SCO	Iceland
819	S-1	ISO8859-1	IS	354	is_IS	SCO	Iceland
819	S-1	ISO8859-1	IS	354	-	Sun	Iceland
1252	S-1	1252	IS	354	-	WIN	Iceland
1275	S-1	1275	IS	354	-	Mac	Iceland
871	S-1	IBM-871	IS	354	-	HOST	Iceland
1149	S-1	IBM-1149	IS	354	-	HOST	Iceland

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
437	S-1	IBM-437	IE	353	-	OS2	Ireland
850	S-1	IBM-850	IE	353	-	OS2	Ireland
819	S-1	ISO8859-1	IE	353	en_IE	AIX	Ireland
850	S-1	IBM-850	IE	353	En_IE	AIX	Ireland
819	S-1	iso88591	IE	353	-	HP	Ireland
1051	S-1	roman8	IE	353	-	HP	Ireland
819	S-1	ISO8859-1	IE	353	en_IE	Sun	Ireland
819	S-1	ISO8859-1	IE	353	en_IE.ISO8859-1	SCO	Ireland
1252	S-1	1252	IE	353	-	WIN	Ireland
1275	S-1	1275	IE	353	-	Mac	Ireland
285	S-1	IBM-285	IE	353	-	HOST	Ireland
1146	S-1	IBM-1146	IE	353	-	HOST	Ireland
862	S-8	IBM-862	IL	972	-	OS2	Israel
916	S-8	ISO8859-8	IL	972	iw_IL	AIX	Israel
1255	S-8	1255	IL	972	-	WIN	Israel
424	S-8	IBM-424	IL	972	-	HOST	Israel
437	S-1	IBM-437	IT	39	-	OS2	Italy
850	S-1	IBM-850	IT	39	-	OS2	Italy
819	S-1	ISO8859-1	IT	39	it_IT	AIX	Italy
850	S-1	IBM-850	IT	39	It_IT	AIX	Italy
819	S-1	iso88591	IT	39	it_IT.iso88591	HP	Italy
1051	S-1	roman8	IT	39	it_IT.roman8	HP	Italy
819	S-1	ISO8859-1	IT	39	it	SCO	Italy
819	S-1	ISO8859-1	IT	39	it_IT	SCO	Italy
819	S-1	ISO8859-1	IT	39	it	Sun	Italy
1252	S-1	1252	IT	39	-	WIN	Italy
1275	S-1	1275	IT	39	-	Mac	Italy
280	S-1	IBM-280	IT	39	-	HOST	Italy
1144	S-1	IBM-1144	IT	39	-	HOST	Italy

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
932	D-1	IBM-932	JP	81	-	OS2	Japan
942	D-1	IBM-942	JP	81	-	OS2	Japan
943	D-1	IBM-943	JP	81	-	OS2	Japan
954	D-1	IBM-eucJP	JP	81	ja_JP	AIX	Japan
932	D-1	IBM-932	JP	81	Ja_JP	AIX	Japan
954	D-1	eucJP	JP	81	ja_JP.eucJP	HP	Japan
5039	D-1	SJIS	JP	81	ja_JP.SJIS	HP	Japan
954	D-1	eucJP	JP	81	ja	SCO	Japan
954	D-1	eucJP	JP	81	ja_JP	SCO	Japan
954	D-1	eucJP	JP	81	ja_JP.EUC	SCO	Japan
954	D-1	eucJP	JP	81	ja_JP.eucJP	SCO	Japan
954	D-1	eucJP	JP	81	ja japanese	Sun	Japan
943	D-1	IBM-943	JP	81	-	WIN	Japan
930	D-1	IBM-930	JP	81	-	HOST	Japan
939	D-1	IBM-939	JP	81	-	HOST	Japan
5026	D-1	IBM-5026	JP	81	-	HOST	Japan
5035	D-1	IBM-5035	JP	81	-	HOST	Japan
949	D-3	IBM-949	KR	82	-	OS2	Korea, South
970	D-3	IBM-eucKR	KR	82	ko_KR	AIX	Korea, South
970	D-3	eucKR	KR	82	ko_KR.eucKR	HP	Korea, South
970	D-3	eucKR	KR	82	ko_KR.eucKR	SGI	Korea, South
970	D-3	5601	KR	82	ko korean	Sun	Korea, South
1363	D-3	1363	KR	82	-	WIN	Korea, South
933	D-3	IBM-933	KR	82	-	HOST	Korea, South
1364	D-3	IBM-1364	KR	82	-	HOST	Korea, South
437	S-1	IBM-437	Lat	3	-	OS2	Latin America
850	S-1	IBM-850	Lat	3	-	OS2	Latin America
819	S-1	ISO8859-1	Lat	3	-	AIX	Latin America
850	S-1	IBM-850	Lat	3	-	AIX	Latin America
819	S-1	iso88591	Lat	3	-	HP	Latin America
819	S-1	ISO8859-1	Lat	3	-	Sun	Latin America
1051	S-1	roman8	Lat	3	-	HP	Latin America
1252	S-1	1252	Lat	3	-	WIN	Latin America
1275	S-1	1275	Lat	3	-	Mac	Latin America
284	S-1	IBM-284	Lat	3	-	HOST	Latin America
1145	S-1	IBM-1145	Lat	3	-	HOST	Latin America
921	S-10	IBM-921	LV	371	-	OS2	Latvia
921	S-10	IBM-921	LV	371	Lv_LV	AIX	Latvia
921	S-10	IBM-921	LV	371	-	WIN	Latvia
1112	S-10	IBM-1112	LV	371	-	HOST	Latvia

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
921	S-10	IBM-921	LT	370	-	OS2	Lithuania
921	S-10	IBM-921	LT	370	Lt_LT	AIX	Lithuania
921	S-10	IBM-921	LV	370	-	WIN	Lithuania
1112	S-10	IBM-1112	LV	370	-	HOST	Lithuania
437	S-1	IBM-437	NL	31	-	OS2	Netherlands
850	S-1	IBM-850	NL	31	-	OS2	Netherlands
819	S-1	ISO8859-1	NL	31	nl_NL	AIX	Netherlands
850	S-1	IBM-850	NL	31	Nl_NL	AIX	Netherlands
819	S-1	iso88591	NL	31	nl_NL.iso88591	HP	Netherlands
1051	S-1	roman8	NL	31	nl_NL.roman8	HP	Netherlands
819	S-1	ISO8859-1	NL	31	nl	SCO	Netherlands
819	S-1	ISO8859-1	NL	31	nl_NL	SCO	Netherlands
819	S-1	ISO8859-1	NL	31	nl	Sun	Netherlands
1252	S-1	1252	NL	31	-	WIN	Netherlands
1275	S-1	1275	NL	31	-	Mac	Netherlands
37	S-1	IBM-037	NL	31	-	HOST	Netherlands
1140	S-1	IBM-1140	NL	31	-	HOST	Netherlands
850	S-1	IBM-850	NZ	64	-	OS2	New Zealand
850	S-1	IBM-850	NZ	64	En_NZ	AIX	New Zealand
819	S-1	ISO8859-1	NZ	64	en_NZ	AIX	New Zealand
819	S-1	ISO8859-1	NZ	64	-	HP	New Zealand
819	S-1	ISO8859-1	NZ	64	en_NZ	SCO	New Zealand
819	S-1	ISO8859-1	NZ	64	en_NZ	Sun	New Zealand
1252	S-1	1252	NZ	64	-	WIN	New Zealand
37	S-1	IBM-037	NZ	64	-	HOST	New Zealand
1140	S-1	IBM-1140	NZ	64	-	HOST	New Zealand
850	S-1	IBM-850	NO	47	-	OS2	Norway
819	S-1	ISO8859-1	NO	47	no_NO	AIX	Norway
850	S-1	IBM-850	NO	47	No_NO	AIX	Norway
819	S-1	iso88591	NO	47	no_NO.iso88591	HP	Norway
1051	S-1	roman8	NO	47	no_NO.roman8	HP	Norway
819	S-1	ISO8859-1	NO	47	no	SCO	Norway
819	S-1	ISO8859-1	NO	47	no_NO	SCO	Norway
819	S-1	ISO8859-1	NO	47	no	Sun	Norway
1252	S-1	1252	NO	47	-	WIN	Norway
1275	S-1	1275	NO	47	-	Mac	Norway
277	S-1	IBM-277	NO	47	-	HOST	Norway
1142	S-1	IBM-1142	NO	47	-	HOST	Norway

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
852	S-2	IBM-852	PL	48	-	OS2	Poland
912	S-2	ISO8859-2	PL	48	pl_PL	AIX	Poland
912	S-2	iso88592	PL	48	pl_PL.iso88592	HP	Poland
912	S-2	ISO8859-2	PL	48	pl_PL.ISO8859-2	SCO	Poland
1250	S-2	1250	PL	48	-	WIN	Poland
1282	S-2	1282	PL	48	-	Mac	Poland
870	S-2	IBM-870	PL	48	-	HOST	Poland
860	S-1	IBM-860	PT	351	-	OS2	Portugal
850	S-1	IBM-850	PT	351	-	OS2	Portugal
819	S-1	ISO8859-1	PT	351	pt_PT	AIX	Portugal
850	S-1	IBM-850	PT	351	Pt_PT	AIX	Portugal
819	S-1	iso88591	PT	351	pt_PT.iso88591	HP	Portugal
1051	S-1	roman8	PT	351	pt_PT.roman8	HP	Portugal
819	S-1	ISO8859-1	PT	351	pt	SCO	Portugal
819	S-1	ISO8859-1	PT	351	pt_PT	SCO	Portugal
819	S-1	ISO8859-1	PT	351	pt	Sun	Portugal
1252	S-1	1252	PT	351	-	WIN	Portugal
1275	S-1	1275	PT	351	-	Mac	Portugal
37	S-1	IBM-037	PT	351	-	HOST	Portugal
1140	S-1	IBM-1140	PT	351	-	HOST	Portugal
852	S-2	IBM-852	RO	40	-	OS2	Romania
912	S-2	ISO8859-2	RO	40	ro_RO	AIX	Romania
912	S-2	iso88592	RO	40	ro_RO.iso88592	HP	Romania
912	S-2	ISO8859-2	RO	40	ro_RO.ISO8859-2	SCO	Romania
1250	S-2	1250	RO	40	-	WIN	Romania
1282	S-2	1282	RO	40	-	Mac	Romania
870	S-2	IBM-870	RO	40	-	HOST	Romania
866	S-5	IBM-866	RU	7	-	OS2	Russia
915	S-5	ISO8859-5	RU	7	-	OS2	Russia
915	S-5	ISO8859-5	RU	7	ru_RU	AIX	Russia
915	S-5	iso88595	RU	7	ru_RU.iso88595	HP	Russia
915	S-5	ISO8859-5	RU	7	ru_RU.ISO8859-5	SCO	Russia
1251	S-5	1251	RU	7	-	WIN	Russia
1283	S-5	1283	RU	7	-	Mac	Russia
1025	S-5	IBM-1025	RU	7	-	HOST	Russia
855	S-5	IBM-855	SP	381	-	OS2	Serbia/Montenegro
915	S-5	ISO8859-5	SP	381	-	OS2	Serbia/Montenegro
915	S-5	ISO8859-5	SP	381	sr_SP	AIX	Serbia/Montenegro
915	S-5	iso88595	SP	381	-	HP	Serbia/Montenegro
1251	S-5	1251	SP	381	-	WIN	Serbia/Montenegro
1283	S-5	1283	SP	381	-	Mac	Serbia/Montenegro
1025	S-5	IBM-1025	SP	381	-	HOST	Serbia/Montenegro

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
852	S-2	IBM-852	SK	938	-	OS2	Slovakia
912	S-2	ISO8859-2	SK	938	sk_SK	AIX	Slovakia
912	S-2	iso88592	SK	938	sk_SK.iso88592	HP	Slovakia
912	S-2	ISO8859-2	SK	938	sk_SK.ISO8859-2	SCO	Slovakia
1250	S-2	1250	SK	938	-	WIN	Slovakia
1282	S-2	1282	SK	938	-	Mac	Slovakia
870	S-2	IBM-870	SK	938	-	HOST	Slovakia
852	S-2	IBM-852	SI	386	-	OS2	Slovenia
912	S-2	ISO8859-2	SI	386	sl_SI	AIX	Slovenia
912	S-2	iso88592	SI	386	sl_SI.iso88592	HP	Slovenia
912	S-2	ISO8859-2	SI	386	sl_SI.ISO8859-2	SCO	Slovenia
1250	S-2	1250	SI	386	-	WIN	Slovenia
1282	S-2	1282	SI	386	-	Mac	Slovenia
870	S-2	IBM-870	SI	386	-	HOST	Slovenia
437	S-1	IBM-437	ZA	27	-	OS2	South Africa
850	S-1	IBM-850	ZA	27	-	OS2	South Africa
819	S-1	ISO8859-1	ZA	27	en_ZA	AIX	South Africa
850	S-1	IBM-850	ZA	27	En_ZA	AIX	South Africa
819	S-1	iso88591	ZA	27	-	HP	South Africa
1051	S-1	roman8	ZA	27	-	HP	South Africa
819	S-1	ISO8859-1	ZA	27	-	Sun	South Africa
819	S-1	ISO8859-1	ZA	27	en_ZA.ISO8859-1	SCO	South Africa
1252	S-1	1252	ZA	27	-	WIN	South Africa
1275	S-1	1275	ZA	27	-	Mac	South Africa
285	S-1	IBM-285	ZA	27	-	HOST	South Africa
1146	S-1	IBM-1146	ZA	27	-	HOST	South Africa
437	S-1	IBM-437	ES	34	-	OS2	Spain
850	S-1	IBM-850	ES	34	-	OS2	Spain
819	S-1	ISO8859-1	ES	34	es_ES	AIX	Spain
					ca_ES		Spain (Catalan)
850	S-1	IBM-850	ES	34	Es_ES	AIX	Spain
					Ca_ES		Spain (Calalan)
819	S-1	iso88591	ES	34	es_ES.iso88591	HP	Spain
1051	S-1	roman8	ES	34	es_ES.roman8	HP	Spain
819	S-1	ISO8859-1	ES	34	es	Sun	Spain
819	S-1	ISO8859-1	ES	34	es	SCO	Spain
819	S-1	ISO8859-1	ES	34	es_ES	SCO	Spain
1252	S-1	1252	ES	34	-	WIN	Spain
1275	S-1	1275	ES	34	-	Mac	Spain
284	S-1	IBM-284	ES	34	-	HOST	Spain
1145	S-1	IBM-1145	ES	34	-	HOST	Spain

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
437	S-1	IBM-437	SE	46	-	OS2	Sweden
850	S-1	IBM-850	SE	46	-	OS2	Sweden
819	S-1	ISO8859-1	SE	46	sv_SE	AIX	Sweden
850	S-1	IBM-850	SE	46	Sv_SE	AIX	Sweden
819	S-1	iso88591	SE	46	sv_SE.iso88591	HP	Sweden
1051	S-1	roman8	SE	46	sv_SE.roman8	HP	Sweden
819	S-1	ISO8859-1	SE	46	sv	SCO	Sweden
819	S-1	ISO8859-1	SE	46	sv_SE	SCO	Sweden
819	S-1	ISO8859-1	SE	46	sv	Sun	Sweden
1252	S-1	1252	SE	46	-	WIN	Sweden
1275	S-1	1275	SE	46	-	Mac	Sweden
278	S-1	IBM-278	SE	46	-	HOST	Sweden
1143	S-1	IBM-1143	SE	46	-	HOST	Sweden
437	S-1	IBM-437	CH	41	-	OS2	Switzerland
850	S-1	IBM-850	CH	41	-	OS2	Switzerland
819	S-1	ISO8859-1	CH	41	de_CH	AIX	Switzerland
850	S-1	IBM-850	CH	41	De_CH	AIX	Switzerland
819	S-1	iso88591	CH	41	-	HP	Switzerland
1051	S-1	roman8	CH	41	-	HP	Switzerland
819	S-1	ISO8859-1	CH	41	de_CH	SCO	Switzerland
819	S-1	ISO8859-1	CH	41	fr_CH	SCO	Switzerland
819	S-1	ISO8859-1	CH	41	it_CH	SCO	Switzerland
819	S-1	ISO8859-1	CH	41	de_CH	Sun	Switzerland
1252	S-1	1252	CH	41	-	WIN	Switzerland
1275	S-1	1275	CH	41	-	Mac	Switzerland
500	S-1	IBM-500	CH	41	-	HOST	Switzerland
1148	S-1	IBM-1148	CH	41	-	HOST	Switzerland
938	D-2	IBM-938	TW	88	-	OS2	Taiwan
948	D-2	IBM-948	TW	88	-	OS2	Taiwan
950	D-2	big5	TW	88	-	OS2	Taiwan
950	D-2	big5	TW	88	Zh_TW	AIX	Taiwan
964	D-2	IBM-eucTW	TW	88	zh_TW	AIX	Taiwan
950	D-2	big5	TW	88	zh_TW.big5	HP	Taiwan
964	D-2	eucTW	TW	88	zh_TW.eucTW	HP	Taiwan
950	D-2	big5	TW	88	big5 zh_TW.big5	Sun	Taiwan
964	D-2	cns11643	TW	88	zh_TW tchinese	Sun	Taiwan
950	D-2	big5	TW	88	-	WIN	Taiwan
937	D-2	IBM-937	TW	88	-	HOST	Taiwan
874	S-20	TIS620-1	TH	66	-	OS2	Thailand
874	S-20	TIS620-1	TH	66	Th_TH	AIX	Thailand
874	S-20	tis620	TH	66	th_TH.tis620	HP	Thailand
874	S-20	TIS620-1	TH	66	-	WIN	Thailand
838	S-20	IBM-838	TH	66	-	HOST	Thailand

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name
857	S-9	IBM-857	TR	90	-	OS2	Turkey
920	S-9	ISO8859-9	TR	90	tr_TR	AIX	Turkey
920	S-9	iso88599	TR	90	tr_TR.iso88599	HP	Turkey
920	S-9	ISO8859-9	TR	90	tr_TR.ISO8859-9	SCO	Turkey
1254	S-9	1254	TR	90	-	WIN	Turkey
1281	S-9	1281	TR	90	-	Mac	Turkey
1026	S-9	IBM-1026	TR	90	-	HOST	Turkey
437	S-1	IBM-437	GB	44	-	OS2	U.K.
850	S-1	IBM-850	GB	44	-	OS2	U.K.
819	S-1	ISO8859-1	GB	44	en_GB	AIX	U.K.
850	S-1	IBM-850	GB	44	En_GB	AIX	U.K.
819	S-1	iso88591	GB	44	en_GB.iso88591	HP	U.K.
1051	S-1	roman8	GB	44	en_GB.roman8	HP	U.K.
819	S-1	ISO8859-1	GB	44	en_UK	Sun	U.K.
819	S-1	ISO8859-1	GB	44	en_GB	SCO	U.K.
819	S-1	ISO8859-1	GB	44	en	SCO	U.K.
1252	S-1	1252	GB	44	-	WIN	U.K.
1275	S-1	1275	GB	44	-	Mac	U.K.
285	S-1	IBM-285	GB	44	-	HOST	U.K.
1146	S-1	IBM-1146	GB	44	-	HOST	U.K.
819	S-1	88591	GB	44	En_GB.88591	SINIX	U.K.
819	S-1	ISO8859-1	GB	44	En_GB.6937	SINIX	U.K.
NOTE: DB2 supports as ISO 8859-1; it should be ISO 6937							
1125	S-12	IBM-1125	UA	380	-	OS2	Ukraine
1124	S-12	IBM-1124	UA	380	uk_UA	AIX	Ukraine
1251	S-12	1251	UA	380	-	WIN	Ukraine
1123	S-12	IBM-1123	UA	380	-	HOST	Ukraine
437	S-1	IBM-437	US	1	-	OS2	USA
850	S-1	IBM-850	US	1	-	OS2	USA
819	S-1	ISO8859-1	US	1	en_US	AIX	USA
850	S-1	IBM-850	US	1	En_US	AIX	USA
819	S-1	iso88591	US	1	en_US.iso88591	HP	USA
1051	S-1	roman8	US	1	en_US.roman8	HP	USA
819	S-1	ISO8859-1	US	1	en_US	Sun	USA
819	S-1	ISO8859-1	US	1	en_US	SGI	USA
819	S-1	ISO8859-1	US	1	en_US	SCO	USA
1252	S-1	1252	US	1	-	WIN	USA
1275	S-1	1275	US	1	-	Mac	USA
37	S-1	IBM-037	US	1	-	HOST	USA
1140	S-1	IBM-1140	US	1	-	HOST	USA
1163	S-13	IBM-1163	VN	84	-	OS2	Vietnam
1163	S-13	IBM-1163	VN	84	vi_VN	AIX	Vietnam
1258	S-13	1258	VN	84	-	WIN	Vietnam
1164	S-13	IBM-1164	VN	84	-	HOST	Vietnam

Table 45. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country Code	Locale	OS	Country Name

Note:

The following map to Arabic Countries (AA):

```
    /* Arabic (Saudi Arabia) */
    /* Arabic (Iraq) */
    /* Arabic (Egypt) */
    /* Arabic (Libya) */
    /* Arabic (Algeria) */
    /* Arabic (Morocco) */
    /* Arabic (Tunisia) */
    /* Arabic (Oman) */
    /* Arabic (Yemen) */
    /* Arabic (Syria) */
    /* Arabic (Jordan) */
    /* Arabic (Lebanon) */
    /* Arabic (Kuwait) */
    /* Arabic (United Arab Emirates) */
    /* Arabic (Bahrain) */
    /* Arabic (Qatar) */
```

The following map to English (US):

```
    /* English (Jamaica) */
    /* English (Carribean) */
```

The following map to Latin America (Lat):

```
   /* Spanish (Mexican) */
   /* Spanish (Guatemala) */
   /* Spanish (Costa Rica) */
   /* Spanish (Panama) */
   /* Spanish (Dominican Republic) */
   /* Spanish (Venezuela) */
   /* Spanish (Colombia) */
   /* Spanish (Peru) */
   /* Spanish (Argentina) */
   /* Spanish (Ecuador) */
   /* Spanish (Chile) */
   /* Spanish (Uruguay) */
   /* Spanish (Paraguay) */
   /* Spanish (Bolivia) */
```

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

Note: Euro-symbol support is provided with this version of DB2 UDB. Microsoft Windows ANSI code pages are modified according to the latest definition from Microsoft to include the Euro-symbol in position 0x80. This position was previously undefined. In addition, the definition of code page 850 has changed to replace the character Dotless i (found at position 0xD5) with the Euro-symbol. DB2 UDB uses the new definitions of these code pages as the default to provide Euro-symbol support. This implementation is the appropriate default for current DB2 UDB customers who require Euro-symbol support, and should not impact other customers. However, if you would like to continue to use the previous definition of these code pages, you may copy the following files:

- 12520850.cnv
- 08501252.cnv
- IBM00850.ucs
- IBM01252.ucs

from this directory

```
sqllib/conv/alt/
```

to this directory

```
sqllib/conv/
```

after installation is complete. You may want to backup the existing `IBM01252.usc` and `IBM00850.ucs` before copying the non-euro versions over them. After copying the files you will not have the euro currency symbol support from DB2 UDB.

Unicode/UCS-2 and UTF-8 Support in DB2 UDB

These two standards are documented here.

Introduction

The Unicode character encoding standard is a fixed-length, character encoding scheme that includes characters from almost all the living languages of the world. Unicode characters are usually shown as “U+xxxx” where xxxx is the hexadecimal code of the character.

Each character is 16 bits (2 bytes) wide, regardless of the language. While the resulting 65 000 code elements are sufficient for encoding most of the characters of the major languages of the world, the Unicode standard also provides an extension mechanism, that allows for encoding as many as a million more characters. This extension reserves a range of code values (U+D800 to U+D8FF, known as “surrogates”) for encoding some 32-bit characters as two successive code elements.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) 10646 standard (ISO / IEC 10646) specifies the Universal Multiple-Octet Coded Character Set (UCS) that has a 2-byte version (UCS-2) and a 4-byte version (UCS-4). The 2-byte version of this ISO standard UCS-2 is identical to Unicode without surrogates. ISO 10646 also defines an extension technique, for encoding some UCS-4 codes in a UCS-2 encoded string. This extension called UTF-16, is identical to Unicode with surrogates.

DB2 UDB supports UCS-2, that is, Unicode without surrogates.

UTF-8

With UCS-2 or Unicode encoding, ASCII and control characters are also two bytes long, and the lead byte is zero. For example, NULL is U+0000 and CAPITAL LETTER A is represented by U+0041. This could be a major problem for ASCII-based applications and ASCII file systems, because in a UCS-2 strings, extraneous NULLs may appear anywhere in the string. A transformation algorithm, known as UTF-8, can be used to circumvent this problem for programs that rely on ASCII code being invariant.

UTF-8 (UCS Transformation Format 8), is an algorithmic transformation which transforms fixed-length UCS-4 characters into variable-length byte strings. In UTF-8, ASCII characters are represented by their usual single-byte codes, but non-ASCII characters in UCS-2 become two or three bytes long. In other words, UTF-8 transforms UCS-2 characters to a multi-byte codeset, for which ASCII is invariant. The number of bytes for each UCS-2 character in UTF-8 format can be determined from the following table:

```
UCS-2 (hex)     UTF-8 (binary)                Description
-----------     --------------------------    ---------------
0000 to 007F    0xxxxxxx                      ASCII
0080 to 07FF    110xxxxx 10xxxxxx             up to U+07FF
0800 to FFFF    1110xxxx 10xxxxxx 10xxxxxx    other UCS-2

NOTE: The range D800 to DFFF is to be excluded from treatment
      by the third row of this table which governs the UCS-4 range
      0000 0800 to 0000 FFFF.
```

In all the above, a series of x's indicate the UCS bit representation of the character. For example, U0080 transforms into 11000010 10000000.

UCS-2/UTF-8 Implementation in DB2 UDB

Code Page/CCSID Numbers

Within IBM, the UCS-2 code page has been registered as code page 1200. All code pages are defined with growing character sets, that is, when new characters are added to a code page, the code page number does not change. Code page 1200 always refers to the current version of Unicode/UCS-2, and has been used for UCS-2 support in DB2 UDB.

A specific repertoire of the UCS standard, as defined by Unicode 2.0 and ISO/IEC 10646-1, has also been registered within IBM as CCSID 13488. This CCSID (13488) has been used internally by DB2 UDB for storing graphic string data in euc-Japan and euc-Taiwan databases. CCSID 13488 and code page 1200 both refer to UCS-2, and are handled the same way except for the value of their "double-byte" (DBCS) space:

CP/CCSID	Single Byte (SBCS) space	Double Byte (DBCS) space
1200	N/A	U+0020
13488	N/A	U+3000

NOTE: In a UCS-2 database, U+3000 has no special meaning.

Regarding the conversion tables, since code page 1200 is a superset of CCSID 13488, the exact same (superset) tables are used for both.

In IBM, UTF-8 has been registered as CCSID 1208 with growing character set (sometimes also referred to as code page 1208). As new characters are added to the standard, this number (1208) will not change either. 1208 is used as the multi-byte code page number for DB2's UCS-2/UTF-8 support.

DB2 UDB supports UCS-2 as a new multi-byte code page. The MBCS code page number is 1208, which is the database code page number, and the code page of character string data within the database. The double-byte code page number (for UCS-2) is 1200 which is the code page of graphic string data within the database. When a database is created in UCS-2/UTF-8, CHAR, VARCHAR, LONG VARCHAR, and CLOB data, are stored in UTF-8, and GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, and DBCLOB data, are stored in UCS-2. We will simply refer to this as a UCS-2 database.

Creating a UCS-2 Database

By default, databases are created in the code page of the application creating them. Therefore, if you create your database from a UTF-8 client (for example,

the UNIVERSAL locale of AIX), or if DB2CODEPAGE environment variable on the client is set to 1208, your database will be created as a UCS-2 database. Alternatively, you can explicitly specify “UTF-8” as the CODESET name, and use any valid two letter TERRITORY code supported by DB2 UDB.

For example, to create a UCS-2 database from the CLP, with the territory code for United States, issue:

```
DB2 CREATE DATABASE dbname USING CODESET UTF-8 TERRITORY US
```

To create a UCS-2 database using the *sqlecrea* API, you should set the values in *sqledbcountryinfo* accordingly. For example, set *sqledbcountryinfo* to “UTF-8”, and *sqldblocale* to any valid territory code (for example, “US”).

The default collation sequence for a UCS-2 database is IDENTITY, which provides UCS-2 code point order. Therefore, by default, all UCS-2/UTF-8 characters are ordered and compared according to their UCS-2 code point sequence.

All cultural-sensitive parameters such as date/time format, decimal separator, and others, are based on the current territory of the client.

A UCS-2 database allows connection from every single-byte and multi-byte code page supported by DB2 UDB. Code page character conversions between client’s code page and UTF-8 are automatically performed by the database manager. Data in graphic string types, is always in UCS-2 and does not go through code page conversions. The Command Line Processor (CLP) environment is an exception. If you SELECT graphic string (UCS-2) data from the CLP, the returned graphic string data is converted (by the CLP) from UCS-2 to the code page of your client environment.

Every client is limited by the character repertoire, the input methods, and the fonts supported by its environment, but the UCS-2 database itself accepts and stores all UCS-2 characters. Therefore, every client usually works with a subset of UCS-2 characters, but the database manager allows the entire repertoire of UCS-2 characters.

When characters are converted from a local code page to UTF-8, there is a possibility of expansion in the number of bytes. There is no expansion for ASCII characters, but other UCS-2 characters expand by a factor of two or three. The number of bytes of each UCS-2 character in UTF-8 format can be determined from the above table (section about UTF-8).

Data Types

All data types supported by DB2 UDB, are also supported in a UCS-2 database. In particular, graphic string data, is supported for UCS-2 database

and is stored in UCS-2/Unicode. Every client, including SBCS clients, can work with graphic string data types in UCS-2/Unicode when connected to a UCS-2 database.

A UCS-2 database is like any MBCS database where character string data is measured in number of bytes. When working with character string data in UTF-8, one should not assume that each character is one byte. In multi-byte UTF-8 encoding, each ASCII character is one byte, but non-ASCII characters take two or three bytes each. This should be taken into account when defining CHAR fields. Depending on the ratio of ASCII to non-ASCII characters, a CHAR field of size n bytes, can contain anywhere from n/3 to n characters.

Using character string UTF-8 encoding versus graphic string UCS-2 data type also has an impact on the total storage requirements. For a situation where the majority of characters are ASCII, with some non-ASCII characters in between, storing UTF-8 data may be a better alternative because the storage requirements are closer to one byte per character. On the other hand, for situations where the majority of characters are non-ASCII characters that expand to three-byte UTF-8 sequences (for example ideographic characters), the UCS-2 graphic-string format may be a better alternative because every UCS-2 character requires exactly two bytes, rather than three bytes for each corresponding character in UTF-8 format.

SQL scalar functions that operate on character strings, such as LENGTH, SUBSTR, POSSTR, MAX, MIN, and the like, in MBCS environments operate on number of "bytes" rather than number of "characters". The behavior is the same in a UCS-2 database but you should take extra care when specifying offsets and lengths for a USC-2 database because these values are always defined in the context of the database code page. That is, in the case of UCS-2 database should be defined in UTF-8. Since some single-byte characters require more than one byte in UTF-8, SUBSTR indexes that are valid for a single-byte database may not be valid for a UCS-2 database. If you specify incorrect indexes, you will get SQLCODE -191, SQLSTATE 22504. Refer to the *SQL Reference* for a description of the behavior of these functions.

SQL CHAR data types are supported by C language's `char` data type in user programs. SQL GRAPHIC data types are supported by `sqldbchar` in user C programs. Note that, for a UCS-2 database, `sqldbchar` data is always in big-endian (high byte first) format. When an application program is connected to a UCS-2 database, character string data is converted between the application code page and UTF-8 by DB2 UDB, but graphic string data is always in UCS-2.

Identifiers

In a UCS-2 database, all identifiers are in multi-byte UTF-8. Therefore, it is possible to use any UCS-2 character in identifiers where the use of a character in the extended character set (for example, an accented character, or a multi-byte character) is allowed by DB2 UDB. Please refer to the appendix "Naming Rules" in the *Administration Guide* for details of which identifiers allow use of extended characters.

Clients can enter any character which is supported by their SBCS/MBCS environment, and all the characters in the identifiers will be converted to UTF-8 by the database manager. Two points need to be taken into account when specifying National Language characters in identifiers in a UCS-2 database:

1. Each non-ASCII character takes two or three bytes. Therefore, an n-byte identifier, can only hold somewhere between n/3 and n characters, depending the ratio of ASCII to non-ASCII characters. If you have only one or two non-ASCII (for example, accented) characters, the limit is closer to n characters, while for an identifier which is completely non-ASCII (for example, in Japanese), only n/3 characters can be used.
2. If identifiers are to be entered from different client environments, they should be defined using the common subset of characters available to those clients. For example, if a UCS-2 database is to be accessed from Latin-1, Arabic, and Japanese environments, all identifiers should realistically be limited to ASCII.

UCS-2 Literals

UCS-2 literals can be specified in two ways:

1. As a GRAPHIC string constant using the G'...', or N'....' format as described in the *SQL Reference*, Chapter 3 "Language Elements", the section "Constants", the subsection "Graphic String Constants." Any literal specified in this way will be converted by the database manager from the application code page to UCS-2.
2. As a UCS-2 hexadecimal string, using the UX'....' or GX'....' format. The constant specified between quotes after UX or GX must be a multiple of 4 hexadecimal digits. Each four digits represent one UCS-2 code point.

When using the Command Line Processor (CLP), the first method is easier if the UCS-2 character exists in the local application code page (for example, for entering any code page 850 character from a terminal that is using code page 850). The second method should be used for characters which are outside the application code page repertoire (for example, for specifying Japanese characters from a terminal that is using code page 850).

Pattern Matching in a UCS-2 Database

Pattern matching is one area where the behavior of existing MBCS databases is slightly different from the behavior of a UCS-2 database.

For MBCS databases in DB2 UDB, the current behavior is as follows: If the match-expression contains MBCS data, the pattern can include both SBCS and MBCS characters. The special characters in the pattern are interpreted as follows:

- An SBCS underscore refers to one SBCS character.
- A DBCS underscore refers to one MBCS character.
- A percent (either SBCS or DBCS) refers to a string of zero or more SBCS or MBCS characters.

If the match-expression contains graphic string DBCS data, the expressions contain only DBCS characters. The special characters in the pattern are interpreted as follows:

- A DBCS underscore refers to one DBCS character.
- A DBCS percent refers to a string of zero or more DBCS characters.

In a UCS-2 database, there is really no distinction between “single-byte” and “double-byte” characters; every UCS-2 character occupies two bytes. Although the UTF-8 format is a “mixed-byte” encoding of UCS-2 characters, there is no real distinction between SBCS and MBCS characters in UTF-8. Every character is a UCS-2 character, irrespective of its number of bytes in UTF-8 format. When specifying a character string, or a graphic string expression, an underscore refers to one UCS-2 characters and a percent refers to a string of zero or more UCS-2 characters.

On the client side, the character string expressions are in the code page of the client, and will be converted to UTF-8 by the database manager. SBCS client code pages, do not have any DBCS percent or DBCS underscore, but every supported code page contains a single-byte percent (corresponding to U+0025) and a single-byte underscore (corresponding to U+005F). The interpretation of special characters for a UCS-2 database is as follows:

- An SBCS underscore (corresponding to U+0025) refers to one UCS-2 character in a graphic string expression, or to one UTF-8 character in a character string expression.
- An SBCS percent (corresponding to U+005F) refers to a string of zero or more UCS-2 characters in a graphic string expression, or to a string of zero or more UTF-8 characters in a character string expression.

DBCS code pages, additionally support a DBCS percent sign (corresponding to U+FF05) and a DBCS underscore (corresponding to U+FF3F). These characters have no special meaning for a UCS-2 database.

For the optional “escape-expression” which specifies a character to be used to modify the special meaning of the underscore and percent characters, only ASCII characters, or characters that expand into a two-byte UTF-8 sequence, are supported. If you specify an ESCAPE character which expands to a three-byte UTF-8 value, you will get an error message (SQL0130N error, SQLSTATE 22019).

IMPORT/EXPORT/LOAD Considerations

The DEL, ASC, and PC/IXF file formats are supported for a UCS-2 database as described in this section. The WSF format is not supported.

When exporting from a UCS-2 database to an ASCII delimited (DEL) file, all character data is converted to the application code page. Both character string and GRAPHIC string data are converted to the same SBCS or MBCS code page of the client. This is the existing behavior for the export of any database, and cannot be changed because the entire ASCII delimited file can have only one code page. Therefore, if you export to an ASCII delimited file, only those UCS-2 characters that exist in your application code page would be saved. Other characters are replaced with the default substitution character for the application code page. For UTF-8 clients (code page 1208) there is no data loss because all UCS-2 characters are supported by UTF-8 clients.

When importing from an ASCII file (DEL or ASC) to a UCS-2 database, character string data is converted from the application code page to UTF-8, and GRAPHIC string data is converted from the application code page to UCS-2. There is no data loss. If you want to import ASCII data that has been saved under a different code page, you should switch the data file code page before issuing the import command. One way to accomplish this, is to set DB2CODEPAGE to the code page of the ASCII data file.

The range of valid ASCII delimiters, for SBCS and MBCS clients is identical to what is currently supported by DB2 UDB for these clients. The range of valid delimiters for UTF-8 clients is 0x01 to 0x7F, with the usual restrictions. Refer to the “IMPORT/EXPORT/LOAD Utility File Formats” appendix in the *Command Reference* for a complete list of these restrictions.

When exporting from a UCS-2 database to a PC/IXF file, character string data is converted to the SBCS/MBCS code page of the client. GRAPHIC string data is not converted and is stored in UCS-2 (code page 1200). There is no data loss.

When importing from an PC/IXF file to a UCS-2 database, character string data is assumed to be in the SBCS/MBCS code page stored in the PC/IXF header and GRAPHIC string data is assumed to be in the DBCS code page stored in the PC/IXF header. Character string data is converted by the IMPORT utility from the code page specified in the PC/IXF header to the code page of the client, and then from the client code page to UTF-8 (by the INSERT statement). GRAPHIC string data is converted by the IMPORT utility from the DBCS code page specified in the PC/IXF header directly to UCS-2 (code page 1200).

LOAD directly places the data into the database and by default, assumes data in ASC or DEL files is in the code page of the database. Therefore, by default no code page conversion takes place for ASCII files. When the code page of the data file has been explicitly specified (using the MODIFIED BY codepage=x command parameter), LOAD uses this information to convert from the specified code page x into the database code page before loading the data. For PC/IXF files, LOAD always converts from the code pages specified in the IXF header to the database code page (1208 for CHAR, 1200 for GRAPHIC).

The code page of DBCLOB files (as specified using the MODIFIED BY lobsinfile command parameter) is always 1200 for UCS-2. The code page of the CLOB files is the same as the code page of the data files being imported, loaded or exported. For example, for load or import using PC/IXF format, the CLOB file is assumed to be in the code page specified by the PC/IXF header. If the DBCLOB file is in ASC or DEL format, for LOAD the CLOB data is assumed to be in the code page of the database (unless explicitly specified otherwise using the MODIFIED BY codepage=x command parameter), and for IMPORT it is assumed to be in the client application code page.

The NOCHECKLENGTHS option is always set to TRUE for a UCS-2 database because in a UCS-2 database, any SBCS can be connected to the database for which there is no DBCS code pages; and also because character strings in UTF-8 format usually have different length than corresponding length in client code pages.

Refer to *Data Movement Utilities Guide and Reference* for more information on the LOAD, EXPORT, and IMPORT utilities.

Incompatibilities

An application connected to a UCS-2 database, the graphic string data is always in UCS-2 (code page 1200). For applications connected to non UCS-2 databases, the graphic string data is in the applications DBCS code page; or, not allowed if the application code page is SBCS. For example, when a 932 client is connected to a Japanese non UCS-2 database, then the graphic string

data is in code page 301. But for the 932 client applications connected to a UCS-2 database, the graphic string data is in UCS-2.

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 Single-Byte Character Set (SBCS) or Double-Byte Character Set (DBCS) code page allows for both single- and double-byte character code points. This is usually accomplished by reserving a subset of the 256 available code points of a mixed code table 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 46. Mixed Character Set Code Points

Country	Supported Mixed Code Page	Code Points for Single-byte Characters	Code Points for First Byte of Double-Byte Characters
Japan	932, 943	x00-7F, xA1-DF	x81-9F, xE0-FC
Japan	942	x00-80, xA0-DF, xFD-FF	x81-9F, xE0-FC
Taiwan	938 (*)	x00-7E	x81-FC
Taiwan	948 (*)	x00-80, FD, FE	x81-FC
Korea	949	x00-7F	x8F-FE
Taiwan	950	x00-7E	x81-FE
China	1381	x00-7F	x8C-FE
Korea	1363	x00-7F	x81-FE
China	1386	x00	x81-FE

Notes:

1. (*) means that this is an old code page and is not recommended anymore.

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 table, there are 256 code points available as the second byte for each valid first byte. Second byte values can have any

value from 0x40 to 0x7E and from 0x80 to 0xFE. Note that in DBCS environments, DB2 does not perform validity checking on individual double-byte characters.

Extended UNIX Code (EUC) Character Sets

Each EUC code page allows for both single-byte character code points, and up to three different sets of multi-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. The remainder of the code points is undefined, allocated as an element of a multi-byte character, or allocated as a single-shift introducer of a multi-byte character. These code points are shown in the following tables.

Table 47. Japanese EUC Code Points

Group	1st Byte	2nd Byte	3rd Byte	4th Byte
G0	x20-7E	n/a	n/a	n/a
G1	xA1-FE	xA1-FE	n/a	n/a
G2	x8E	xA1-FE	n/a	n/a
G3	x8E	xA1-FE	xA1-FE	n/a

Table 48. Traditional Chinese EUC Code Points

Group	1st Byte	2nd Byte	3rd Byte	4th Byte
G0	x20-7E	n/a	n/a	n/a
G1	xA1-FE	xA1-FE	n/a	n/a
G2	x8E	xA1-FE	xA1-FE	xA1-FE
G3	n/a	n/a	n/a	n/a

Table 49. Korean EUC Code Points

Group	1st Byte	2nd Byte	3rd Byte	4th Byte
G0	x20-7E	n/a	n/a	n/a
G1	xA1-FE	xA1-FE	n/a	n/a
G2	n/a	n/a	n/a	n/a
G3	n/a	n/a	n/a	n/a

Table 50. Simplified Chinese EUC Code Points

Group	1st Byte	2nd Byte	3rd Byte	4th Byte
G0	x20-7E	n/a	n/a	n/a
G1	xA1-FE	xA1-FE	n/a	n/a
G2	n/a	n/a	n/a	n/a
G3	n/a	n/a	n/a	n/a

Code points not assigned to the categories shown above are not defined, and are treated as single-byte undefined code points.

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 also 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.

Bidirectional CCSID Support

The following BiDi attributes are required for correct handling of Bidirectional data on different platforms:

- Text type (LOGICAL vs VISUAL)
- Shaping (SHAPED vs UNSHAPED)
- Orientation (RIGHT-TO-LEFT vs LEFT-TO-RIGHT)
- Numeral shape (ARABIC vs HINDI)
- Symmetric swapping (YES or NO)

Defaults on different platforms are not the same, problems appear when DB2 data is sent from one platform to another. For example, Windows platforms use LOGICAL UNSHAPED data, while data on OS/390 is usually in SHAPED VISUAL format. Therefore, without any support for bidirectional attributes, data sent from DB2 Universal Database for OS/390 to DB2 UDB on a Windows 32-bit operating systems may display incorrectly.

Bidirectional-specific CCSIDs

DB2 supports bidirectional data attributes through special bidirectional Coded Character Set Identifiers (CCSIDs). The following bidirectional CCSIDs have been defined and are implemented with DB2 UDB:

CCSID	Code Page	String Type
00420	420	4
00424	424	4
08612	420	5
08616	424	6
62208	856	4
62209	862	4
62210	916	4
62211	424	5
00856	856	5
62213	862	5
00916	916	5
01255	1255	5
01046	1046	5
00864	864	5

01089	1089	5
01256	1256	5
62220	856	6
62221	862	6
62222	916	6
62223	1255	6
62224	420	6
62225	864	6
62226	1046	6
62227	1089	6
62228	1256	6
62235	424	10
62236	856	10
00862	862	10
62238	916	10
62239	1255	10
62240	424	11
62241	856	11
62242	862	11
62243	916	11
62244	1255	11

Where CDRA String Types are defined:

String Type	Text Type	Numerical Shape	Orientation	Shaping	Symmetrical Swapping
4	Visual	Arabic	LTR	Shaped	OFF
5	Implicit	Arabic	LTR	Unshaped	ON
6	Implicit	Arabic	RTL	Unshaped	ON
7(*)	Visual	Arabic	Contextual(*)	Unshaped-Lig	OFF
8	Visual	Arabic	RTL	Shaped	OFF
9	Visual	Passthru	RTL	Shaped	ON
10	Implicit		Contextual-L		ON
11	Implicit		Contextual-R		ON

Note: (*) Field orientation is left-to-right (LTR) when the first alphabetic character is a Latin one, and right-to-left (RTL) when it is a bidirectional (RTL) character. Characters are unshaped, but LamAlef ligatures are kept, and not broken into constituents.

DB2 Universal Database Implementation of Bidirectional Support

Bidirectional layout transformations are implemented in DB2 Universal Database using the new CCSID definitions. For the new BiDi-specific CCSIDs, layout transformations are performed instead of or in addition to code page conversions. To use this support, the DB2BIDI registry variable must be set to YES. By default, this variable is not set. This variable is used by the server for all conversions, and can only be set when the server is started. Setting DB2BIDI to YES may have some performance impact because of additional checking and layout transformations.

To specify a specific bidirectional CCSID in non-DRDA environment, select the appropriate CCSID from the above table that matches the characteristics of your client, and set DB2CODEPAGE to that value. If you already have a connection to the database, you must issue a TERMINATE command and connect again to make the new setting of DB2CODEPAGE take effect. If you select a CCSID which is not correct for code page or string type of your client platform, results would be unexpected. If you select an incompatible CCSID (i.e., Hebrew CCSID for connection to an Arabic database or vice-versa), or if DB2BIDI has not been set for the server, you will receive an error message when you try to connect.

For DRDA environments, if the HOST EBCDIC platform also supports these bidirectional CCSIDs, you need to only set DB2CODEPAGE as mentioned above. However, if HOST platform does not support these CCSIDs, you must specify a CCSID override for the HOST database server that you are connecting to. This is necessary because, in DRDA environment, code page conversions and layout transformations are performed by the receiver of data. However, if HOST server does not support these bidirectional CCSIDs, it does not perform layout transformation on the data that it receives from DB2 UDB. If you use a CCSID override, the DB2 UDB client performs layout transformation on the outbound data as well. For details of how to set a CCSID override, please refer to DB2 Connect Release Notes.

CCSID override is not supported for cases where the HOST EBCDIC platform is the client and DB2 UDB is the server.

DB2 Connect Implementation of Bidirectional Support

When data is exchanged between DB2 Connect and a database on a server, it is usually the receiver that performs conversion on the incoming data. The same convention would normally apply to bidirectional layout transformations also, which is in addition to the usual code page conversion. DB2 Connect has the optional ability to perform bidirectional layout transformation on data it is about to send to the server database in addition to data received from the server database.

In order for DB2 Connect to perform bidirectional layout transformation on outgoing data for a server database, the bidirectional Coded Character Set Identifier (CCSID) of the server database will have to be overridden. This is accomplished through the use of the BIDI parameter in the PARMS field of the DCS database directory entry for the server database.

Note: If you would like DB2 Connect to perform layout transformation on the data it is about to send to the DB2 host database, even though you do not have to override its CCSID, you still have to add the BIDI

parameter in the DCS database directory PARMS field. In this case, the CCSID that you should provide is the default DB2 host database CCSID.

The BIDI parameter is to be specified as the ninth parameter in the PARMS field along with the bidirectional CCSID with which the user would like to override the default server database bidirectional CCSID in the following format:

```
",,,,,,,,BIDI=xyz"
```

where xyz is the CCSID override.

Note: The registry variable, DB2BIDI, must be set to "YES" in order for the BIDI parameter to take effect.

A list of the bidirectional CCSIDs that are supported along with their string types is found in "Bidirectional-specific CCSIDs" on page 773.

The use of this feature is best illustrated with an example.

Suppose you have a Hebrew DB2 client running CCSID 62213 (bidirectional string type 5) and you would like to access a DB2 host database running CCSID 00424 (bidirectional string type 4). However, you know that the data contained in the DB2 host database is instead based on CCSID 08616 (bidirectional string type 6).

There are two problems in this situation: The first is that the DB2 host database does not know the difference in the bidirectional string types with CCSIDs 00424 and 08616. The second problem is that the DB2 host database does not recognize the DB2 client CCSID of 62213. It only supports CCSID 00862, which is based on the same code page as CCSID 62213.

You will need to make sure that data sent to the DB2 host database is in bidirectional string type 6 format to begin with and also let DB2 Connect know that it has to perform bidirectional transformation on data it receives from the DB2 host database. You will need to use following catalog command for the DB2 host database:

```
db2 catalog dcs database nydb1 as telaviv parms ",,,,,,,,BIDI=08616"
```

What this command does is tell DB2 Connect to override the DB2 host database CCSID of 00424 with 08616. This override includes the following processing:

1. DB2 Connect connects to the DB2 host database using CCSID 00862.

2. DB2 Connect performs bidirectional layout transformation on the data it is about to **send** to the DB2 host database. The transformation is from CCSID 62213 (bidirectional string type 5) to CCSID 62221 (bidirectional string type 6).
3. DB2 Connect performs bidirectional layout transformation on data it **receives** from the DB2 host database. The transformation is from CCSID 08616 (bidirectional string type 6) to CCSID 62213 (bidirectional string type 5).

Note: In some cases, use of a bidirectional CCSID may cause the SQL query itself to be modified such that it is not recognized by the DB2 server. Specifically, you should avoid using IMPLICIT CONTEXTUAL and IMPLICIT RIGHT-TO-LEFT CCSIDs when a different string type can be used. CONTEXTUAL CCSIDs can produce unpredictable results if the SQL query contains quoted strings. Avoid using quoted strings in SQL statements and use host variables instead whenever possible.

If a specific bidirectional CCSID is causing problems which cannot be rectified by following these recommendations, then you should set DB2BIDI=NO.

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; the assignments of numbers to characters in a set are collectively called a *code page*. 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', &ellipsis; 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, the collation weights for X'82', X'60', and X'61' are used to sort these two characters according to their code points.

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 all cases, DB2 uses the collation table which was specified at database creation time. If you require the multi-byte characters to sort the way they appear in their code point table, you must specify IDENTITY as your collation sequence when you create the database.

Note: For DBCS characters in GRAPHIC fields, the sort sequence is always IDENTITY without regard to the collation sequence specified at database creation time.

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 *Administrative 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'
```

Collating Sequence Sort Order: EBCDIC and ASCII Example

The order in which data in a database is sorted depends on the collating sequence defined for the database. For example, suppose that database A uses the EBCDIC code page's default collating sequence and that database B uses the ASCII code page's default collating sequence. Sort orders at these two databases would differ, as shown in Figure 77 on page 781.

```
SELECT.....
  ORDER BY COL2

EBCDIC-Based Sort        ASCII-Based Sort

COL2                     COL2
----                     ----
V1G                      7AB
Y2W                      V1G
7AB                      Y2W
```

Figure 77. Example of How a Sort Order in an EBCDIC-Based Sequence Differs from a Sort Order in an ASCII-Based Sequence

Similarly, character comparisons in a database depend on the collating sequence defined for that database. So if database A uses the EBCDIC code page's default collating sequence and database B uses the ASCII code page's default collating sequence, the results of character comparisons at the two databases would differ. Figure 78 illustrates the difference.

```
SELECT.....
  WHERE COL2 > 'TT3'

EBCDIC-Based Results     ASCII-Based Results

COL2                     COL2
----                     ----
TW4                      TW4
X72                      X72
39G
```

Figure 78. Example of How a Comparison of Characters in an EBCDIC-Based Sequence Differs from a Comparison of Characters in an ASCII-Based Sequence

If you are creating a federated database, consider specifying that your collating sequence matches the collating sequence at a data source. This approach will maximize "pushdown" opportunities and possibly increase query performance. See the *Administration Guide, Performance* for more information on the relationship between pushdown analysis, collating sequences, and query performance.

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 code point 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.

General 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*.

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 code points of characters will only match the results of a query ordered using an identity collating sequence.

Federated Database Concerns: Your choice of database collating sequence can affect federated system performance. If a data source uses the same collating sequence as the DB2 federated database, DB2 can pushdown order-dependent processing involving character data to the data source. If a data source collating sequence does not match DB2's, data is retrieved and all order-dependent processing on character data is done locally (which can slow performance).

To determine if a data source and DB2 have the same collating sequence, consider the following factors:

- National language support

  The collating sequence is related to the language supported on a server. Compare DB2 NLS information to data source NLS information.
- Data source characteristics

  Some data sources are created using case-insensitive collating sequences, which can yield different results from DB2 in order-dependent operations.
- Customization

  Some data sources provide multiple options for collating sequences or allow the collating sequence to be customized.

Choose the collating sequence for a DB2 federated database based on the mix of data sources that will be accessed from that database. For example:

- If a DB2 database will access mostly Oracle databases with the same code page (NLS) as DB2, specify SQL_CS_NONE for the SQLDBCSS structure (Oracle databases use an equivalent collating sequence).

- If a DB2 database will access only DB2 UDB databases, ensure that you match SQLDBCSS values.

For information on how to set up a MVS collating sequence, you should refer to the*Administrative API Reference* for samples under the “sqlecrea” topic. These samples contain collation tables for EBCIDIC 500, 37, and 5026/5035 code pages.

After you set the collating sequence for the DB2 database, ensure that you set the *collating_sequence* server option for each data source server. The collating_sequence option indicates if the collating sequence of a given data source server matches the collating sequence of the DB2 database.

Set the collating_sequence option to “Y” if the collating sequences match. This setting allows the DB2 optimizer to consider order-dependent processing at a data source, which can improve performance. However, if the data source collating sequence is not the same as the DB2 database collating sequence, you can receive incorrect results. For example, if your plan uses merge joins, the DB2 optimizer will pushdown ordering operations to the data sources as much as possible. If the data source collating sequence is not the same, the join results may not have a correct result set.

Set the collating_sequence option to “N” if the collating sequences do not match. Use this value when data source collating sequences differ from DB2 or when the data source collating operations might be case insensitive. For example, in a case insensitive data source with an English code page, TOLLESON, ToLLeSoN, and tolleson would all be considered equal. Set the collating_sequence option to “N” if you are not sure that the collating sequence at the data source is identical to the DB2 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 53 on page 788 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 51. 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 52. Each format is identified by name and includes an associated abbreviation and an example of its use.

Table 52. 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

Table 52. Formats for String Representations of Times (continued)

Format Name	Abbreviation	Time Format	Example
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 53.
- Output Date Format
  - The default output date format is shown in Table 53.

    Note: Table 53 also shows a listing of the string formats for the various country codes.

Table 53. 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

Table 53. Date and Time Formats by Country Code (continued)

Country Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
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
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

Table 53. Date and Time Formats by Country Code (continued)

Country Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
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 I. Issuing Commands to Multiple Database Partition Servers

In a partitioned database system, you may want to issue commands to be run on machines in the instance, or on database partition servers (nodes). You can do so using the **rah** command or the **db2_all** command. The **rah** command allows you to issue commands that you want to run at machines in the instance. If you want the commands to run at database partition servers in the instance, you run the **db2_all** command. This section provides an overview of these commands. The information that follows applies to partitioned database systems only.

Notes:

1. On UNIX-based platforms, your login shell can be a Korn shell or any other shell; however, there are differences in the way the different shells handle commands containing special characters.
2. On Windows NT, to run the **rah** command or the **db2_all** command, you must be logged on with a user account that is a member of the Administrators group.

To determine the scope of a command, refer to the *Command Reference*. This book indicates whether a command runs on a single database partition server, or on all of them. If the command runs on one database partition server and you want it to run on all of them, use **db2_all**. The exception is the **db2trc** command, which runs on all the logical nodes (database partition servers) on a machine. If you want to run **db2trc** on all logical nodes on all machines, use **rah**.

Commands

You can run the commands sequentially at one database partition server after another, or you can run the commands in parallel. On UNIX-based platforms, 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 behavior) or the output can be displayed at the machine where the command is issued. On Windows NT, if you run the commands in parallel, the output is displayed at the machine where the command is issued.

To use the **rah** command, type:

```
rah command
```

To use the **db2_all** command, type:

```
db2_all command
```

To obtain help about **rah** syntax, type

```
rah "?"
```

The command can be almost anything which you could type at an interactive prompt, including, for example, multiple commands to be run in sequence. On UNIX-based platforms, you separate multiple commands using a semicolon (;). On Windows NT, you separate multiple commands using an ampersand (&). Do not use the separator character following the last command.

The following example shows how to use the **db2_all** command to change the database configuration on all database partition servers that are specified in the node configuration file. Because the ; character is placed inside double quotation marks, the request will run concurrently:

```
db2_all ";UPDATE DB CFG FOR sample USING LOGFILSIZ=100"
```

Command Descriptions

You can use the following commands:

Command	Description
rah	Runs the command on all machines.
db2_all	Runs the command on all database partition servers that you specify.
db2_kill	Abruptly stops all processes being run on multiple database partition servers and cleans up all resources on all database partition servers. This command renders your databases inconsistent. Do *not* issue this command except under direction from IBM service.
db2_call_stack	On UNIX-based platforms, causes all processes running on all database partition servers to write call traceback to the syslog. On Windows NT, causes all processes running on all database partition servers to write call traceback to the P*xxxx*.*nnn* file in the instance directory, where P*xxxx* is the process ID and *nnn* is the node number.

On UNIX-based platforms, these commands execute **rah** with certain implicit settings such as:

- Run in parallel at all machines

- Buffer command output in /tmp/$USER/db2_kill, /tmp/$USER/db2_call_stack respectively.

On Windows NT, these commands execute **rah** to run in parallel at all machines.

Specifying the Command to Run

You can 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 $
```

If you specify the command as the parameter on the command line, you must enclose it in double quotation marks if it contains any of the special characters just listed.

Note: On UNIX-based platforms, the command will be added to your command history just as if you typed it at the prompt.

All special characters in the command can be entered normally (without being enclosed in quotation marks, except for \). If you need to include a \ in your command, you must type two backslashes (\ \).

Note: On UNIX-based platforms, if you are not using a Korn shell, all special characters in the command can be entered normally (without being enclosed in quotation marks, except for " \ unsubstituted $, and the single quotation mark (')). 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 quotation mark (") in your command, you must precede it by three backslashes, for example, \ \ \".

Notes:

1. On UNIX-based platforms, You cannot include a single quotation mark (') in your command unless your command shell provides some way of entering a single quotation mark inside a singly quoted string.
2. On Windows NT, you cannot include a single quotation mark (') in your command unless your command window provides some way of entering a single quotation mark inside a singly quoted string.

Running Commands in Parallel on UNIX-Based Platforms

Note: The information in this section applies to UNIX-based platforms only. By default, the command is run sequentially at each machine, 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 machine, 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 prefixes 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 machines, ensure that:

- A directory `/tmp/$USER` exists for your user ID at each machine. To create a directory if one does not already exist, run:

  ```
  rah ")mkdir /tmp/$USER"
  ```

- Add the following line to your .kshrc (for Korn shell syntax) or .profile, and also type it into your current session:

  ```
  export RAHCHECKBUF=no
  ```

- Ensure that each machine 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 machine ID at which you run the remote command.

Monitoring rah Processes on UNIX-Based Platforms

Note: The information in this section applies to UNIX-based platforms only. 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. Refer to the help information for details 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 the 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 the name of the symbolic link). If you want, 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 the 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 Korn shell (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 Korn shell, then the login shell at the ID which executes **rah** must also not be a Korn shell. (**rah** makes the decision as to whether the remote ID's shell is a Korn shell based on the local ID). The shell must not perform any substitution or special processing on a string enclosed in single quotation marks. It must leave it exactly as is.

Additional Rah (Run All Hosts) Information (Solaris and AIX only)

To enhance performance, rah has been extended to use tree_logic on large systems. That is, rah will check how many nodes the list contains, and if that number exceeds a threshold value, it constructs a subset of the list and sends a recursive invocation of itself to those nodes. At those nodes, the recursively invoked rah follows the same logic until the list is small enough to follow the standard logic (now the ″leaf-of-tree″ logic) of sending the command to all nodes on the list. The threshold can be specified by environment variable RAHTREETHRESH, or defaults to 15.

In the case of a multiple-logical-node-per-physical-node system, db2_all will favor sending the recursive invocation to distinct physical nodes, which will

then rsh to other logical nodes on the same physical node, thus also reducing inter-physical-node traffic. (This point applies only to db2_all, not rah, since rah always sends only to distinct physical nodes.)

This version of rah has nearly identical syntax and semantics as the old version (supplied in the product as rah.sh_old), except for some minor restrictions on what options can be used:

- The user must use ksh (kornshell) as the shell. If the user is using a different shell, rah_tree will issue a warning and use non-tree logic.
- The single quotation mark (') prefix character, requesting echo of the command, cannot be honored, and is ignored.
- When both the < (all-but-me) and > (substitute <> by host index) options are specified, the host index is different from what non-tree rah / db2_all would have substituted.

  Note: The () and ## substitutions should work identically to the old rah / db2_all.
- It is strongly recommended that the user ID setup at all nodes in the list be identical; for example, the current working directory from which the rah_tree command or the db2_tree command is issued should exist on all nodes; the rah_tree executable must be found in the current $PATH on all nodes (the one in effect when the rah_tree command is issued); the rahwaitfor executable must be found in that path, and so on. Certain environment differences between nodes can be tolerated, but many can not.
- The command to be executed must not start with the characters `-o`, `-b`, `-d`, or `-x`, because rah_tree will interpret these to be flags.
- When specifying parallel execution, the order in which hosts return their output is likely to be different from the order generated by non-tree rah, which tends to return output in list order.
- Whenever rah_tree or db2_tree is invoked, it compares the number of destination nodes with a threshold value, as described above. The threshold can be specified by environment variable:

  ```
  export RAHTREETHRESH=nn
     where nn can be any positive integer
  ```

  or defaults to 15.

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 multicharacter sequence

must be typed in order. If you type any prefix sequences, you must enclose the entire command, including the prefix sequences in double quotation marks, as in the following examples:

- On UNIX-based platforms:

```
rah "};ps -F pid,ppid,etime,args -u $USER"
```

- On Windows NT:

```
rah "||db2 get db cfg for sample"
```

The prefix sequences are:

Sequence	Purpose
|	Runs the commands in sequence in the background.
|&	Runs the commands in sequence in the background and terminates the command after all remote commands have completed, even if some are still running. This may be later if, for example, child processes (on UNIX-based platforms) or background processes (on Windows NT) are still running. In this case, the command starts a separate background process to retrieve any remote output generated after command termination and writes it back to the originating machine. **Note:** On UNIX-based platforms, specifying & degrades performance, because more **rsh** commands are required.
||	Runs the commands in parallel in the background.
||&	Runs the commands in parallel in the background and terminates the command after all remote commands have completed as described for the |& case above. **Note:** On UNIX-based platforms, specifying & degrades performance, because more **rsh** commands are required.
;	Same as ||& above. This is an alternative shorter form. **Note:** On UNIX-based platforms, specifying ; degrades performance relative to ||, because more **rsh** commands are required.
]	Prepends dot-execution of user's profile before executing command. **Note:** Available on UNIX-based platforms only.
}	Prepends dot-execution of file named in $RAHENV (probably .kshrc) before executing command. **Note:** Available on UNIX-based platforms only.

]} Prepends dot-execution of user's profile followed by execution of file named in $RAHENV (probably .kshrc) before executing command.

Note: Available on UNIX-based platforms only.

) Suppresses execution of user's profile and of file named in $RAHENV.

Note: Available on UNIX-based platforms only.

' Echoes the command invocation to the machine.

< Sends to all the machines except this one.

<<−nnn<
Sends to all-but-database partition server *nnn* (all database partition servers in db2nodes.cfg except for node number *nnn*, see the note below).

<<+nnn<
Sends to only database partition server *nnn* (the database partition server in db2nodes.cfg whose node number is *nnn*, see the note below).

Runs the remote command in the background with `stdin`, `stdout` and `stderr` all closed. This option is valid only when running the command in the background, that is, only in a prefix sequence which also includes | or ;. It allows the command 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 quotation marks, or enclose the command in double quotation marks, and precede the by \. For example,

```
rah ';mydaemon'
```

or

```
rah ";\ mydaemon"
```

When run as a background process, the **rah** command will never wait for any output to be returned.

> Substitutes occurrences of <> with the machine name.

" Substitutes occurrences of () by the machine index, and substitutes occurrences of ## by the node number.

Notes:

1. The machine index is a number that associated with a machine in the database system. If you are not running multiple logical nodes, the machine index for a machine corresponds to the node number for that machine in the node configuration file. To obtain the machine index for a machine in a multiple logical node

environment, do not count duplicate entries for those machines that run multiple logical nodes. For example, if MACH1 is running two logical nodes and MACH2 is also running two logical nodes, the node number for MACH3 is 5 in the node configuration file. The machine index for MACH3, however, would be 3.

On Windows NT, do not edit the node configuration file. To obtain the machine index, use the **db2nlist** command. Refer to the *DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings* manual for details.

2. When " is specified, duplicates are not eliminated from the list of machines. See "Eliminating Duplicate Entries from the List of Machines" on page 800 if you want to eliminate duplicates.

When using the <<-nnn< and <<+nnn< prefix sequences, *nnn* is any 1-, 2- or 3-digit partition number which must match the *nodenum* value in the db2nodes.cfg file.

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 quotation marks.

Specifying the List of Machines

By default, the list of machines is taken from the node configuration file, db2nodes.cfg. You can override this by:

- Specifying a pathname to the file that contains the list of machines by exporting (on UNIX-based platforms) or setting (on Windows NT) the environment variable RAHOSTFILE.
- Specifying the list explicitly, as a string of names separated by spaces, by exporting (on UNIX-based platforms) or setting (on Windows NT) the environment variable RAHOSTLIST.

  Note: If both of these environment variables are specified, RAHOSTLIST takes precedence.

Note: On Windows NT, to avoid introducing inconsistencies into the node configuration file, do *not* edit it manually. To obtain the list of machines in the instance, use the **db2nlist** command. Refer to the *DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings* manual for details.

Eliminating Duplicate Entries from the List of Machines

If you are running DB2 Enterprise - Extended Edition with multiple logical nodes (database partition servers) on one machine, your db2nodes.cfg file will contain multiple entries for that machine. In this situation, the **rah** command needs to know whether you want the command to be executed once only on each machine or once for each logical node listed in the db2nodes.cfg file. Use the **rah** command to specify machines. Use the **db2_all** command to specify logical nodes.

Note: On UNIX-based platforms, if you specify machines, **rah** will normally eliminate duplicates from the machine list, with the following exception: if you specify logical nodes, **db2_all** prepends the following assignment to your command:

```
export DB2NODE=nnn  (for Korn shell syntax)
```

where *nnn* is the node number taken from the corresponding line in the db2nodes.cfg file, so that the command will be routed to the desired database partition server.

When specifying logical nodes, you can restrict the list to include all logical nodes except one, or only specify one database partition server using the <<-nnn< and <<+nnn< prefix sequences. You may want to do this if you want to run a command at the catalog node first, and when that has completed, run the same command at all other database partition servers, possibly in parallel. This is usually required when running the **db2 restart database** command. You will need to know the node number of the catalog node to do this. See "Prefix Sequences" on page 796 for information about the prefix sequences.

If you execute **db2 restart database** using the **rah** command, duplicate entries are eliminated from the list of machines. However if you specify the " prefix, then duplicates are not eliminated, because it is assumed that use of the " prefix implies sending to each database partition server, rather than to each machine.

Controlling the rah Command

You can use the following environment variables to control the **rah** command.

Table 54.

Name	Meaning	Default
$RAHBUFDIR **Note:** Available on UNIX-based platforms only.	directory for buffer	/tmp/$USER
$RAHBUFNAME **Note:** Available on UNIX-based platforms only.	filename for buffer	rahout
$RAHOSTFILE (on UNIX-based platforms); RAHOSTFILE (on Windows NT)	file containing list of hosts	db2nodes.cfg
$RAHOSTLIST (on UNIX-based platforms); RAHOSTLIST (on Windows NT)	list of hosts as a string	extracted from $RAHOSTFILE
$RAHCHECKBUF **Note:** Available on UNIX-based platforms only.	if set to ″no″, bypass checks	not set
$RAHSLEEPTIME (on UNIX-based platforms); RAHSLEEPTIME (on Windows NT)	time in seconds this script will wait for initial output from commands run in parallel	86400 seconds for **db2_kill**, 200 seconds for all other
$RAHWAITTIME (on UNIX-based platforms); RAHWAITTIME (on Windows NT)	on Windows NT, interval in seconds between successive checks that remote jobs are still running. On UNIX-based platforms, interval in seconds between successive checks that remote jobs are still running and `rah: waiting for <pid> ...` messages. On all platforms, specify any positive integer. Prefix value with a leading zero to suppress messages, for example, export RAHWAITTIME=045. It is not necessary to specify a low value as **rah** does not rely on these checks to detect job completion.	45 seconds

Table 54. (continued)

Name	Meaning	Default
$RAHENV **Note:** Available on UNIX-based platforms only.	specifies filename to be executed if $RAHDOTFILES=E or K or PE or B	$ENV
$RAHUSER (on UNIX-based platforms); RAHUSER (on Windows NT)	on UNIX-based platforms, user ID under which the remote command is to be run. On Windows NT, the logon account associated with the DB2 Remote Command Service	$USER

Note: On UNIX-based platforms, the value of $RAHENV where **rah** is run is used, not the value (if any) set by the remote shell.

$RAHDOTFILES on UNIX-Based Platforms

Note: The information in this section applies to UNIX-based platforms only. 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 a Korn shell, any dot files you specify to be executed will be executed in a Korn shell process, and so must conform to Korn shell syntax. So, for example, if your login shell is a C shell, to have your .cshrc environment set up for commands executed by **rah**, you should either create a Korn shell INSTHOME/.profile equivalent to your .cshrc and specify in your INSTHOME/.cshrc:

```
setenv RAHDOTFILES P
```

or you should create a Korn shell INSTHOME/.kshrc equivalent to your .cshrc and specify in your INSTHOME/.cshrc:

```
setenv RAHDOTFILES E
setenv RAHENV INSTHOME/.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
```

Setting the Default Environment Profile on Windows NT

Note: The information in this section applies to Windows NT only.
To set the default environment profile for the **rah** command, use a file called db2rah.env, which should be created in the instance directory. The file should have the following format:

```
; This is a comment line
DB2INSTANCE=instancename
DB2DBDFT=database
; End of file
```

You can specify all the environment variables that you need to initialize the environment for **rah**.

Determining Problems with rah on UNIX-Based Platforms

Note: The information in this section applies to UNIX-based platforms only.
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** sends a command to all machines to check the existence of the buffer directory, and to create it if it does not exist.
   - One or more of the machines where you are sending your command is not responding. The **rsh** command will eventually time out but the time-out interval is quite long, usually about 60 seconds.

2. You have received messages such as:

   - Login incorrect
   - Permission denied

   Either one of the machines does not have the ID running **rah** correctly defined in its ·hosts file, or the ID running **rah** does not have one of the machines correctly defined in its ·rhosts file.

3. When running commands in parallel using background rshells, although the commands run and complete within the expected elapsed time at the machines, **rah** takes a long time to detect this and put up the shell prompt.

The ID running **rah** does not have one of the machines 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. Instead, use the default (15).

5. The output from **rah** is not displayed correctly, or **rah** incorrectly reports 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 (for example, $RAHBUFDIR/$RAHBUFNAME) for buffering the outputs. To prevent this problem, use a different $RAHBUFNAME for each concurrent **rah** command, for example in the following ksh:

   ```
   export RAHBUFNAME=rahout
   rah ";$command_1" &
   export RAHBUFNAME=rah2out
   rah ";$command_2" &
   ```

   or use a method that 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 file 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 receive an error because the command **print from ()** is not valid.

For a performance tip when running commands in parallel, use | rather than |&, and use || rather than ||& or ; unless you truly need the function provided by &. Specifying & requires more **rsh** commands and therefore degrades performance.

Appendix J. How DB2 for Windows NT Works with Windows NT Security

When you install Windows NT, it allows you to create two administrator usernames:

- One is called “Administrator”
- The other is a name of your choice. It must have administrator authority and must comply with DB2’s naming rules. For more information on DB2’s naming rules, see “Appendix D. Naming Rules” on page 691.

The user may logon to the local machine, or when the machine is installed in a Windows NT Advanced Server Domain, the user may logon to the Domain. DB2 for Windows NT supports both of these options. To authenticate the user, DB2 checks the local machine first, then the Domain Controller for the current Domain, and finally any Trusted Domains known to the Domain Controller.

To illustrate how this works, suppose that the DB2 instance requires Server authentication. The configuration is as follows:

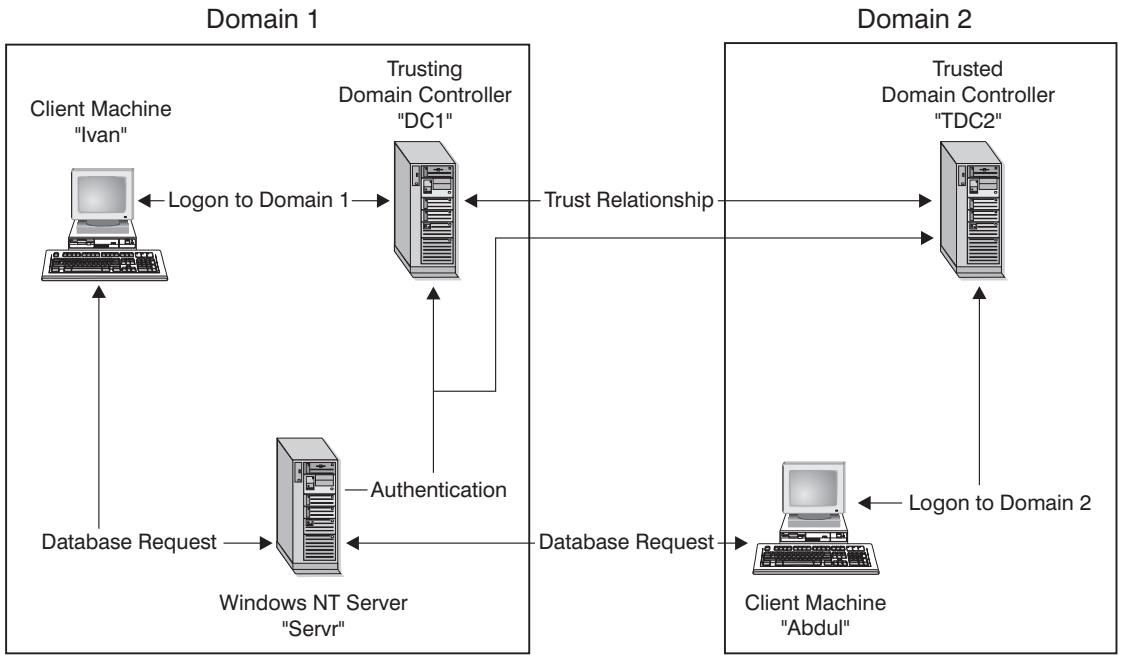


Figure 79. Authentication Using Windows NT Domains

Each machine has a security database, Security Access Management (SAM), unless a client machine is running Windows 95. Windows 95 machines do not have a SAM database. DC1 is the domain controller, in which the client machine, Ivan, and the DB2 for Windows NT server, Servr, are enrolled. TDC2 is a trusted domain for DC1 and the client machine, Abdul, is a member of TDC2's domain.

A Sample Scenario with Server Authentication:

1. Abdul logs on to the TDC2 domain (that is, he is known in the TDC2 SAM database).
2. Abdul then connects to a DB2 database that is cataloged to reside on SRV3:

   ```
   db2 connect to remotedb user Abdul using fredpw
   ```

3. SRV3 determines where Abdul is known. The API that is used to find this information first searches the local machine (SRV3) and then the domain controller (DC1) before trying any trusted domains. Username Abdul is found on TDC2. This search order requires a single namespace for users and groups.
4. SRV3 then:
   a. Validates the username and password with TDC2.
   b. Finds out whether Abdul is an administrator by asking TDC2.
   c. Enumerates all Abdul's groups by asking TDC2.

A Sample Scenario with Client Authentication and a Windows NT Client Machine:

1. Dale, the administrator, logs on to SRV3 and changes the authentication for the database instance to Client:

   ```
   db2stop myinst
   db2 update dbm cfg using authentication client
   db2start myinst
   ```

2. Ivan, at a Windows NT client machine, logs on to the DC1 domain (that is, he is known in the DC1 SAM database).
3. Ivan then connects to a DB2 database that is cataloged to reside on SRV3:

   ```
   DB2 CONNECT to remotedb user Ivan using johnpw
   ```

4. Ivan's machine validates the username and password. The API used to find this information first searches the local machine (Ivan) and then the domain controller (DC1) before trying any trusted domains. Username Ivan is found on DC1.
5. Ivan's machine then validates the username and password with DC1.
6. SRV3 then:
   a. Determines where Ivan is known.

b. Finds out whether Ivan is an administrator by asking DC1.
c. Enumerates all Ivan's groups by asking DC1.

Note: Before attempting to connect to the DB2 database, ensure that DB2 for Window NT Security Service has been started. The Security Service is installed by DB2 and is set up to run as a Windows NT service; however, it is not started automatically. To start the DB2 Security Service, enter the `NET START DB2NTSECSERVER` command.

A Sample Scenario with Client Authentication and a Windows 95 Client Machine:

1. Dale, the administrator, logs on to SRV3 and changes the authentication for the database instance to Client:

   ```
   db2stop myinst
   db2 update dbm cfg using authentication client
   db2start myinst
   ```

2. Ivan, at a Windows 95 client machine, logs on to the DC1 domain (that is, he is known in the DC1 SAM database).
3. Ivan then connects to a DB2 database that is cataloged to reside on SRV3:

   ```
   db2 connect to remotedb user Ivan using johnpw
   ```

4. Ivan's Windows 95 machine cannot validate the username and password. The username and password are therefore assumed to be valid.
5. SRV3 then:
   a. Determines where Ivan is known.
   b. Finds out whether Ivan is an administrator by asking DC1.
   c. Enumerates all Ivan's groups by asking DC1.

Note: Because a Windows 95 client cannot validate a given username and password, client authentication under Windows 95 is inherently insecure. If the Windows 95 machine has access to a Windows NT security provider, however, some measure of security can be imposed by configuring the Windows 95 system for validated pass-through logon. For details on how to configure your Windows 95 system in this way, refer to the Microsoft documentation for Windows 95.

DB2 also supports global groups. In order to use global groups, you must include global groups inside a local group that is on the security server. When DB2 enumerates all the groups that a person is a member of, it also lists the local groups the user is a member of indirectly (by the virtue of being in a global group that is itself a member of one or more local groups).

Using a Backup Domain Controller with DB2

If the server you use for DB2 also acts as a backup domain controller, you can improve DB2 performance and reduce network traffic if you configure DB2 to use the backup domain controller.

You specify the backup domain controller to DB2 by setting the *db2dmnbckctlr* registry value.

If you know the name of the domain for which DB2 server is the backup domain controller, use:

```
db2dmnbckctlr=DOMAIN_NAME
```

where *DOMAIN_NAME* must be in upper case.

To have DB2 determine the domain for which the local machine is a backup domain controller, use:

```
db2dmnbckctlr=?
```

Note: DB2 does not use an existing backup domain controller by default because a backup domain controller can get out-of-sync with the primary domain controller, causing a security exposure. Domain controllers get out-of-sync when the primary domain controller's security database is updated but the changes are not propagated to a backup domain controller. This can happen if there are network latencies or if the computer browser service is not operational.

Appendix K. Using the Windows NT Performance Monitor

There are two performance monitors available to DB2 for Windows NT users:

- **DB2 Performance Monitor**

  The DB2 Performance Monitor provides snapshot and event data related to DB2 and DB2 Connect only. (For more information, click on the **Help** push button in the Control Center and see the Getting Started online help.)

- **Windows NT Performance Monitor**

  The Windows NT Performance Monitor enables you to monitor both database and system performance, retrieving information from any of the performance data providers registered with the system. Windows NT also provides performance information data on all aspects of machine operation including:

  – CPU usage
  – Memory utilization
  – Disk activity
  – Network activity

Registering DB2 with the Windows NT Performance Monitor

The setup program automatically registers DB2 with the Windows NT Performance Monitor for you.

To make DB2 and DB2 Connect performance information accessible to the Windows NT Performance Monitor, you must register the DLL for the DB2 for Windows NT Performance Counters. This also enables any other Windows NT application using the Win32 performance APIs to get performance data.

To install and register the DB2 for Windows NT Performance Counters DLL (DB2Perf.DLL) with the Windows NT Performance Monitor, type:

```
db2perfi -i
```

This copies the DLL to the directory \SYSTEM32 under the system directory. To find the name of the system directory, type:

```
echo %systemroot%
```

Registering the DLL also creates a new key in the services option of the registry. One entry gives the name of the DLL, which provides the counter support. Three other entries give names of functions provided within that DLL. These functions include:

- **Open**

  Called when the DLL is first loaded by the system in a process.
- **Collect**

  Called to request performance information from the DLL.
- **Close**

  Called when the DLL is unloaded.

Enable Remote Access to DB2 Performance Information

If your DB2 for Windows NT workstation is networked to other Windows NT machines, you can use the feature described in this section.

In order to see Windows NT performance objects from another DB2 for Windows NT machine, you must register an administrator username and password with DB2. (The default Windows NT Performance Monitor username, **SYSTEM**, is a DB2 reserved word and cannot be used.) To register the name, type:

```
db2perfr -r username password
```

Note: The `username` used must conform to the naming rules.

The username and password data is held in a key in the registry, with security that allows access only by administrators and the SYSTEM account. The data is encoded to prevent security concerns about storing an administrator password in the registry.

Notes:

1. Once a username and password combination has been registered with DB2, even local instances of the Performance Monitor will explicitly log on using that username and password. This means that if the username information registered with DB2 does not match, local sessions of the Performance Monitor will not show DB2 performance information.
2. The username and password combination must be maintained to match the username and password values stored in the Windows NT Security database. If the username or password is changed in the Windows NT Security database, the username and password combination used for remote performance monitoring must be reset.
3. To deregister, type:

   ```
   db2perfr -u
   ```

Displaying DB2 and DB2 Connect Performance Values

To display DB2 and DB2 Connect performance values using the Performance Monitor, simply choose the performance counters whose values you want displayed from the **Add to** box. This box displays a list of performance objects providing performance data. Select an object to see a list of the counters it supplies.

A performance object can also have multiple instances. For example, the LogicalDisk Object provides counters such as "% Disk Read Time" and "Disk Bytes/sec"; it also has an instance for each logical drive in the machine, including "C:" and "D:".

Windows NT provides the following performance objects:

- **DB2 Database Manager**

  This object provides general information for a single Windows NT instance. The DB2 instance being monitored appears as the object instance.

  For practical and performance reasons, you can only get performance information from one DB2 instance at a time. The DB2 instance that the Performance Monitor shows is governed by the `db2instance` registry variable in the Performance Monitor process. If you have multiple DB2 instances running simultaneously and want to see performance information from more than one, you must start a separate session of the Performance Monitor, with `db2instance` set to the relevant value for each DB2 instance to be monitored.
- **DB2 Databases**

  This object provides information for a particular database. Information is available for each currently active database.
- **DB2 Applications**

  This object provides information for a particular DB2 application. Information is available for each currently active DB2 application.
- **DB2 DCS Databases**

  This object provides information for a particular DCS database. Information is available for each currently active database.
- **DB2 DCS Applications**

  This object provides information for a particular DB2 DCS application. Information is available for each currently active DB2 DCS application.

Which of these objects will be listed by the Windows NT Performance Monitor depends on what is installed on your Windows NT machine and what applications are active. For example, if DB2 UDB is installed and the Database Manager has been started, the DB2 Database Manager object will be listed. If there are also some DB2 databases and applications currently active

on that machine, the DB2 Databases and DB2 Applications objects will be listed as well. If you are using your Windows NT system as a DB2 Connect gateway and there are some DCS databases and applications currently active, the DB2 DCS Databases and DB2 DCS Applications objects will be listed.

Accessing Remote DB2 Performance Information

Enabling remote access to DB2 Performance Information was discussed in an earlier section. In the **Add to** box, select another computer to monitor. This brings up a list of all the available performance objects on that computer.

In order to be able to monitor DB2 Performance object on a remote computer, the level of the DB2 UDB or DB2 Connect code installed on that computer must be Version 6 or higher.

Resetting DB2 Performance Values

When an application calls the DB2 monitor APIs, the information returned is normally the cumulative values since the DB2 server was started. However, often it is useful to:

- Reset performance values
- Run a test
- Reset the values again
- Re-run the test.

To reset database performance values, use the **db2perfc** program. Type:

```
db2perfc
```

By default, this resets performance values for all active DB2 databases. However, you can also specify a list of databases to reset. You can also use the -d option to specify that performance values for DCS databases should be reset. For example:

```
db2perfc
db2perfc dbalias1 dbalias2 ... dbaliasn

db2perfc  -d
db2perfc  -d  dbalias1 dbalias2 ... dbaliasn
```

The first example resets performance values for all active DB2 databases. The next example resets values for specific DB2 databases. The third example resets performance values for all active DB2 DCS databases. The last example resets values for specific DB2 DCS databases.

The program resets the values for ALL programs currently accessing database performance information for the relevant DB2 server instance (that is, the one held in `db2instance` in the session in which you run **db2perfc**.

Invoking **db2perfc** also resets the values seen by anyone remotely accessing DB2 performance information when the **db2perfc** command is executed.

Note: There is a DB2 API, `sqlmrset`, that allows an application to reset the values it sees locally, not globally, for particular databases.

Appendix L. Configuring Multiple Logical Nodes

You can configure multiple logical nodes in one of two ways:

- Configure the logical nodes (database partitions) in the db2nodes.cfg file. You can then start all the logical and remote nodes with the DB2START command or its associated API.
- Restart a logical node on another processor on which other logical database partitions (nodes) are already running. This allows you to override the hostname and port number specified for the logical database partition in db2nodes.cfg.

To configure a logical database partition (node) in db2nodes.cfg, you must make an entry in the file to allocate a logical port number for the node. Following is the syntax you should use:

```
nodenumber hostname logical-port netname
```

Note: You must ensure that you define enough ports in etc/services for FCM communications.

Appendix M. Using Virtual Interface (VI) Architecture

Virtual Interface (VI) Architecture is the inter-node communication protocol alternative to TCP/IP in a Windows NT massively parallel processing (MPP) environment. VI is a new communication architecture that was developed jointly by Intel, Microsoft, and Compaq to improve performance over a System Area Network (SAN). Refer to http://www.viarch.org for more information on the architecture.

Products exist which may be acquired separately from DB2 UDB that have a VIA-enabled network interface card (NIC), switch, and software driver implementation. Several Independent Hardware Vendors (IHVs) have released, or plan to release, such products.

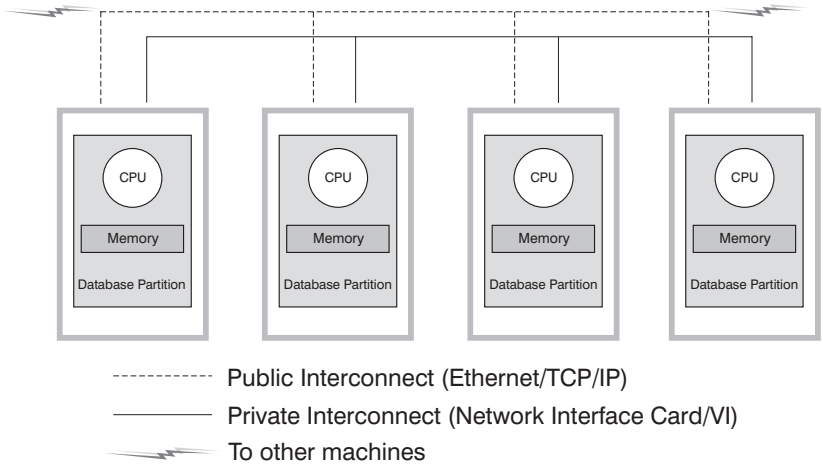


Figure 80. Network Interface Card and Protocol

As shown in Figure 80, there are some similarities between the Public Interconnect which uses as an example Ethernet and TCP/IP and the Private Interconnect which uses a Network Interface Card and a protocol.

The Network Interface Card and protocol used in this instance could be either a GigaNet Network Interface Card and the VI protocol; or a ServerNet Network Interface Card and the VI protocol.

VI Architecture has low latency and high bandwidth. In a communication-intensive environment, using VI Architecture improves the

overall system throughput. The greater the number of nodes in the cluster, and the greater the amount of data transferred, the greater the benefit from using VI Architecture.

DB2 UDB supports VI Architecture implementations that comply with the *Virtual Interface Architecture Specification, Version 1.0*; the *Intel Virtual Interface (VI) Architecture Developers' Guide, Version 1.0*; and pass the "Virtual Interface Architecture Conformance Suite". The specification is found at http://www.intel.com/design/servers/vi/the_spec/specification.htm on the Web. The Developer's Guide is found at http://www.intel.com/design/servers/vi/developer/ia_imp_guide.htm on the Web. Information on the conformance suite is also found at this same URL.

Overview of DB2 UDB Extended Enterprise Edition

DB2 UDB EEE is a relational database management system that enables local and remote client applications to create, update, control, and manage relational databases using Structured Query Language (SQL), ODBC, JDBC, or CLI. Its main feature is the ability for a database to be partitioned across multiple independent machines of a common platform. To the end-user and application developer, it still appears as a single database on a single machine. This fully scalable database system enables an application to use a database that is simply too large for a single machine to handle efficiently. SQL operations and utilities can execute in parallel both within and between the individual database partitions, thereby speeding up the execution time of a single query or command.

IBM announced support for Virtual Interface (VI) Architecture with DB2 UDB EEE V5.2.

To find out about other products adhering to VI Architecture and supported by DB2 UDB EEE, please contact the DB2 UDB support organization at http://www.software.ibm.com/data or call 1-800-237-5511 (only in the U.S.A).

The products that have been tested with DB2 UDB include:

- GigaNet Interconnect, see "Running DB2 UDB for Windows NT with GigaNet Interconnect" on page 821 for details.
- ServerNet Interconnect, see "Running DB2 UDB for Windows NT with ServerNet Interconnect" on page 823 for details.

There may be other products that work with DB2 Universal Database. Check with the vendor of that product, and with IBM Service and Support, to ensure that the other product is supported.

Running DB2 UDB for Windows NT with GigaNet Interconnect

To find out about GigaNet products, or to contact GigaNet Service and Support, please use the following URL: http://www.giganet.com/

Setup Procedure for GigaNet Interconnect

The list of the hardware and software required to setup this environment include the following products:

- GigaNet GNN1000 Network Interface Card
- GigaNet GNX5000 Switch
- GigaNet GNCxx11 Copper Interconnect Cables
- GigaNet cLAN Software, Version 2.0.

The steps required to ensure that GigaNet Interconnect can work with DB2 UDB are shown below. Each step is a summary of what is required at each step: all of the details associated with each step are not presented here. You should also use the referenced documentation at each step which does provide detailed instructions and direction needed.

Each GigaNet GNN1000 is packaged with a GigaNet cLAN Software CD-ROM. The CD-ROM contains all of the necessary software to set-up the GigaNet Interconnect. In addition, the CD-ROM also contains the VI Architecture SDK and Adobe Acrobat Reader. These two items are only needed by those individuals that are developing VI-enabled applications.

Summary of steps:

1. Install Adapter Cards
2. Install Switches and Cables
3. Install Adapter Drivers
4. Install cLAN Management Console
5. Test the Interconnect

Here are the steps:

1. Install the GigaNet GNN1000 Network Interface Card. Please refer to the *GigaNet GNN1000 User Guide* for installation instructions.
2. Install the GigaNet GNX5000 Switch and Cables. Please refer to the *GigaNet GNX5000 User Guide* for installation instructions.
3. Install the GigaNet GNN1000 Adapter Driver software on each node connected to the GNX5000 Switch. Please refer to the *GigaNet GNN1000 User Guide* for installation instructions. Here are additional details if you are installing drivers provided by GigaNet:

a. Remove any previous version of the GNN1000 Driver already installed. Removal requires the node to be re-booted.
b. Use `Start→Setting→Control Panel→Networks→Adapters→Add` to install the driver.
c. Click **Have Disk...** and specify the Driver directory on the CD-ROM. For example, if F: is your CD-ROM drive, then you would use `F:\Driver`
d. Select "GNN1000 NDIS Adapter" and then click **OK**.
e. Configure Network protocols to complete the installation.

GigaNet Adapter Driver software is also available on GigaNet's web site, http://www.giganet.com. Please refer to the download and installation instructions found on the support page of GigaNet's web site.

The installation of the GNN1000 Adapter Driver causes the node to re-boot.

4. The GigaNet cLAN Management Console (GMC) can be used to test the integrity of the GigaNet Interconnect. The GigaNet cLAN Management Console is comprised of two parts: the Console, and the Agent. The Agent must be installed on all nodes in the cluster. The Console can be installed on any network node that has access to the nodes in the cluster. The most versatile and recommended installation is that which has both the Console and the Agent installed on each node in the cluster.

   Install the GigaNet cLAN Management Console. Please refer to the *GigaNet GNN1000 User Guide* for installation instructions and additional information about the cLAN Management Console. Here are additional details on the installation procedure:

   a. Insert the cLAN Software CD into the CD-ROM drive.
   b. Wait for the CD automatic installation menu to appear.
   c. Click on "Install cLAN Management Console."
   d. Repeat this installation procedure on each remaining node in the cluster.

   GigaNet cLAN Management Console software is also available on GigaNet's web site, http://www.giganet.com. Please refer to the download and installation instructions found on the support page of GigaNet's web site.

   The installation of the cLAN Management Console may cause the node to re-boot.

5. Test that the GigaNet Hardware is working. This can be done by doing the following:

   a. Open the GMC. (`Programs→GigaNet→cLAN Management Console`)

b. A dialog box is displayed showing all accessible machines in the LAN. Press **ESC**.

c. Select **Console→Local** from the menu bar.

d. Confirm that all the members in the cluster are shown and that they are all "Active".

e. Select **Utilities→VI Throughput** from the menu bar. This will run a throughput test to check that the data is actually going through the hardware.

f. Enter in uppercase letters the computer names of the two nodes you wish to use in the test. Identify the local node as the source node.

g. Click **Start Measuring**. You should see data being transferred at a rate of at least 65 MB per second.

h. Click **Stop Measuring** to stop the connection test.

i. Repeat the test for the other nodes in the cluster by measuring throughput between the local node (Source) and the other nodes (Sink).

If the connection test does not appear to be working, refer to the troubleshooting sections of the *GigaNet GNN1000 User Guide* and the *GigaNet GNX5000 User Guide*.

See "Install DB2 Universal Database Version 5.2 or Later (EEE)" on page 826 for information on how to install and implement DB2 UDB to work with GigaNet Interconnect.

Running DB2 UDB for Windows NT with ServerNet Interconnect

To find out about ServerNet products, or to contact ServerNet Service and Support, please use the following URL: http://www.servernet.com/

Setup Procedure for ServerNet Interconnect

The list of the hardware and software required to setup this environment include the following products:

- ServerNet PCI Adapter Driver (SPAD), (product ID T0089), version 1.3.5 or later
- ServerNet Switch 1
- ServerNet Area Network Manager (SANMan), (product ID T0087), version 1.1.3 or later.

The following are the steps required to ensure that ServerNet Interconnect can work with DB2 UDB. Each step is a summary of what is required at each step: all of the details associated with each step are not presented here. You

should also use the referenced documentation at each step which does provide detailed instructions and direction needed.

The steps shown below also assume that you are only using up to six (6) nodes in the cluster. Contact ServerNet if you have a requirement to use more than six nodes.

Here are the steps:

1. Install the ServerNet Network Interface Card. Please refer to the *ServerNet-I Virtual Interface Software Release Document, (product ID N0031)* for installation instructions.
2. Install the ServerNet Switch 1. Please refer to the *ServerNet-I Virtual Interface Software Release Document, (product ID N0031)* for installation instructions.
3. Uninstall previous ServerNet drivers. (Skip this step if this is your first time installing ServerNet.)
   a. Open the Network control panel. (`Start→Setting→Control Panel→Network`)
   b. Click on the **Adapters Tab**.
   c. Remove Tandem ServerNet PCI Adapter Driver.
   d. Click on the **Services Tab**.
   e. Remove SANMan.
   f. Click on the **Protocols Tab**.
   g. Remove Tandem ServerNet-I VI Protocol.
4. Install the Tandem ServerNet PCI Adapter Driver. Here are additional details if you are installing using the software CD provided by ServerNet:
   a. Open the Network control panel. (`Start→Setting→Control Panel→Network`)
   b. Click on the **Adapters Tab**. (The Adapters screen appears.)
   c. Ensure the new ServerNet driver is placed in a separate drive and/or directory. Then, from the command prompt referencing the correct drive and/or directory, type “ernnn.exe -d” to start the self-extracting program. (“ernnn.exe” is the name of the Engineering Release followed by a number — ERnnn.EXE — that identifies the specific version of the ServerNet driver to be installed.)
   d. Change to the drive and/or directory where the extracted files are located. Change to the “Spad n.n.n \ Free” sub-directory (where “n.n.n” is the specific version of the product). (If you are working in a troubleshooting or a development environment, then change to the “Spad n.n.n \ Checked” sub-directory instead of the “Spad n.n.n \ Free” sub-directory.)
   e. Rename the “oemsetup.multi_node” file to “oemsetup.inf”.

f. Choose **Add** in the Adapters Tab. (The Select Adapters screen appears.)

g. Click **Have Disk...**. (The Insert Disk screen appears.)

h. Enter the drive and/or directory where the oemsetup.inf file is located.

i. Ensure the dialog box shows "Tandem ServerNet PCI Adapter Driver" and then click **OK**. Ensure the list of adapters now shows the ServerNet adapter. Click **Close**.

j. Choose **Yes** to restart the computer. Or, select **No** and continue installing SANMan and the VI Software Developer's Kit (SDK).

5. Install SANMan. Here are additional details if you are installing using the software CD provided by ServerNet:

   a. Open the Network control panel. (`Start→Setting→Control Panel→Network`)

   b. Click on the **Services Tab**. (The Services screen appears.)

   c. Ensure the new ServerNet driver is placed in a separate drive and/or directory. Then, from the command prompt referencing the correct drive and/or directory, type "ernnn.exe -d" to start the self-extracting program. ("ernnn.exe" is the name of the Engineering Release followed by a number — ERnnn.EXE — that identifies the specific version of the ServerNet driver to be installed.)

   d. Choose **Add** in the Services Tab. (The Select Services screen appears.)

   e. Change to the drive and/or directory where the extracted files are located. Change to the "SANMan n.n.n \Free" sub-directory (where "n.n.n" is the specific version of the product). (If you are working in a troubleshooting or a development environment, then change to the "SANMan n.n.n \ Checked" sub-directory instead of the "SANMan n.n.n \ Free" sub-directory.)

   f. Determine if the Switch is X or Y by looking at the light on the Switch. One light says "X", and the one light says "Y".

   g. If an X Switch, select X=1 and Y=0. Ensure all cables are connected to the X port on the network cards.

   h. If a Y Switch, select X=0 and Y=1. Ensure all cables are connected to the Y port on the network cards.

   i. Provide the port number of the switch to which the network card on the current machine is connected.

   j. Select "PC" for all six (6) ports.

6. Install the Virtual Interface Protocol. Here are additional details if you are installing using the software CD provided by ServerNet:

   a. Open the Network control panel. (`Start→Setting→Control Panel→Network`)

b. Click on the **Protocols Tab**. (The Network Protocols screen appears.)
c. Ensure the new ServerNet driver is placed in a separate drive and/or directory. Then, from the command prompt referencing the correct drive and/or directory, type "ernnn.exe -d" to start the self-extracting program. ("ernnn.exe" is the name of the Engineering Release followed by a number — ERnnn.EXE — that identifies the specific version of the ServerNet driver to be installed.)
d. Choose **Add** in the Protocols Tab. (The Select Network Protocols screen appears.)
e. Click **Have Disk...**. (The Insert Disk screen appears.)
f. Enter the drive and/or directory where the extracted files are located.

7. Test that the ServerNet Hardware is working. There are no test programs available. Instead, simply use DB2 to test the ServerNet hardware.

   If the hardware does not appear to be working, refer to the *ServerNet-I Virtual Interface Software Release Document, (product ID N0031)* for additional troubleshooting help.

See "Install DB2 Universal Database Version 5.2 or Later (EEE)" for information on how to install and implement DB2 UDB to work with ServerNet Interconnect.

Install DB2 Universal Database Version 5.2 or Later (EEE)

Detailed installation information is found in *DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings*.

You will require the first (1st) DB2 UDB Version 5.2 (EEE) FixPak to be able to use DB2 UDB Version 5.2 with a VI product. Later versions of DB2 UDB will not require this FixPak. If you are not sure of the level of the DB2 UDB product you have already installed, you should use the `db2level` command and record the information returned. If you contact DB2 Service and Support about VI, this information will be helpful to determine your installed DB2 level of code including any FixPaks.

This product must be installed in each of the partitions/nodes using the Virtual Interface Protocol. During the installation procedure, when prompted choose "This machine will be an instance owning node" on each of the partitions/nodes.

Update the hosts file with the IP address and host name for each of the partitions/nodes. The hosts file is found under "\winnt\system32\drivers\etc\" directory on the drive where the operating system was installed. The hosts file must be updated on each of the nodes.

Create the partitioned database (MPP) instance using the instance create utility. Choose one machine to act as the coordinator node. On this machine, open a DB2 Command Window and enter:

```
db2icrt <instance_name> /mpp /u:<username>,<password>
```

This machine is then known as the coordinator node or the instance-owning machine. Node 0 is automatically created on this machine.

On the other partitions/nodes in the database, open a DB2 Command Window and enter:

```
db2ncrt /n:<node_number> /u:<username>,<password>
          /i:<instance_name> /o:<instance_owner_name>
```

The node_number is used to uniquely identify the database partition server within the database environment. The number must be from 1 to 999. The instance_owner_name is the computer name of the instance-owning machine (coordinator node).

Testing the installation and create an index:

1. Open a DB2 Command Window.
2. Enter `set DB2INSTANCE=<instance_name>`
3. Ensure the database manager starts on all nodes by entering: `db2start`
4. Create a sample database by entering: `db2sampl`
5. Connect to the sample database by entering: `db2 connect to sample`
6. Try a few SELECT statements with the sample database.

When problems occur in this environment, you can take action based on the type of problem as presented below:

- Instance creation fails.

  Ensure `c:\profiles` is present and is present with each of the share name "profiles". Ensure all partitions are "pingable" from the coordinator node.
- DB2START fails.

  Review the explanation for the returned error code by using the `db2 ? sqlxxxx` command. There will be a suggested action associated with this error which you should follow.

  A system error may be returned. If this is the case, use a `db2stop` and retry the `db2start`. If the problem persists, attempt to reboot on all partitions and then retry.

  Ensure all partitions have the same date, time, and time zone. The time does not need to be identical: within one hour is sufficient.

  Ensure all the partitions are in one domain and that the user name and password used belongs to the following groups:

  - On the domain controller:
    - Administrators
    - Domain Administrators
    - Domain Users
    - Users
  - On other machines:
    - Administrators
    - Users

  Review the contents of the `Control panel->Services` to ensure that all the `DB2:<instance_name> -X` services have the correct DB2ADMIN account information.
- Command line variable has not been initialized:

  Ensure you are running the command in a "DB2 command window". The title of this window is "DB2 CLP".
- The `rah` command returns immediately without executing the commands specified:

  Run `db2set -g DB2TEMPDIR=C:\TMP` on all machines in the instance. Ensure the DB2 Remote Command Service is started and with the correct DB2ADMIN account information. Finally, ensure `c:\temp` and `c:\tmp` are present.

Implement DB2 to Run Using VI

On each database partition server in the instance, set the following DB2 registry variables and carry out the following tasks:

- Set DB2_VI_ENABLE=ON
- Set DB2_VI_DEVICE=nic0
- Set DB2_VI_VIPL=vipl.dll
- Enter `db2start` on the MPP instance.
- Review the db2diag.log file. There should be one message for each partition stating that "VI is enabled."

Appendix N. Lightweight Directory Access Protocol (LDAP) Directory Services

Lightweight Directory Access Protocol (LDAP) is an industry standard access method to directory services. Each instance of the database server will publish its existence and provide the protocol communication information in the LDAP directory. When a client connects to the database server, the communication information for the server can be retrieved from the LDAP directory. Each client is no longer required to store the server connection information by cataloging a node entry locally on each machine. Instead, when a database is created, the database publishes its existence using the LDAP directory. Client applications search the LDAP directory for the database location and the information required to connect to the database.

A caching mechanism exists so that the client only searches the LDAP directory once. Once the information is retrieved, it is stored or cached on the local machine. Subsequent access to the same information is through the cached information and not through another search using the LDAP directory.

Registration of DB2 Servers After Installation

Each DB2 server instance must be registered in LDAP to publish the protocol configuration information that is used by the client applications to connect to it. When registering an instance of the database server, you need to specify a *node name*. The node name is used by client applications when they connect or attach to the server. You can catalog another alias name for the LDAP node by using the CATALOG LDAP NODE command.

The REGISTER command appears as follows:

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
```

The `protocol` clause specifies the communication protocol to use when connecting to this database server.

When creating an instance for DB2 Universal Database EEE that includes multiple physical machines, the REGISTER command must be invoked once for each machine. The *rah* command is used to issue the REGISTER command on all machines.

Note: The same ldap_node_name cannot be used for each machine since the name must be unique in LDAP. You will want to substitute the hostname of each machine for the ldap_node_name in the REGISTER command. For example:

```
rah ">DB2 REGISTER DB2 SERVER IN LDAP AS <> PROTOCOL TCPIP"
```

The "<>" is substituted by the hostname on each machine where the rah command is run. In the rare occurrence where there are multiple DB2 Universal Database EEE instances, the combination of the instance and host index may be used as the node name in the *rah* command.

The REGISTER command can be issued for a remote DB2 server. To do so, you must specify the remote computer name, instance name, and the protocol configuration parameters when registering a remote server. The command can be used as follows:

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
   hostname <host_name>
   svcename <tcpip_service_name>
   remote <remote_computer_name>
   instance <instance_name>
```

The following convention is used for the computer name:

- If TCP/IP is configured, the computer name must be the same as the TCP/IP hostname.
- If APPC is configured, the computer name must be the same as the LU name.

When running in a high availability or fail-over environment, and using TCP/IP as the communication protocol, the *cluster* IP address must be used. Using the cluster IP address allows the client to connect to the server on either machines without having to catalog a separate TCP/IP node for each machine. The cluster IP address is specified using the hostname parameter, shown as follows:

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
   hostname n.nn.nn.nn
```

where n.nn.nn.nn is the cluster IP address.

Refer to the *Command Reference* for additional information on the REGISTER command.

Update the Protocol Information for the DB2 Server

The DB2 server information in LDAP must be kept current. For example, changes to the protocol configuration parameters or the server network address require an update to LDAP.

To update the DB2 server in LDAP on the local machine, use the following command:

```
db2 update db2 server in ldap ...
```

Examples of protocol configuration parameters that can be updated include:

- A TCP/IP hostname and service name of port number parameters.
- A computer name for Named Pipe support.
- An IPX address.
- APPC protocol information like TP name, partner LU, or mode.

To update a remote DB2 server protocol configuration parameters use the UPDATE command with a node parameter:

```
db2 update db2 server in ldap
   node <node_name>
   hostname <host_name>
   svcename <tcpip_service_name>
```

Catalog a Node Alias for ATTACH

A node name for the DB2 server must be specified when registering the server in LDAP. Applications use the node name to attach to the database server. If a different node name is required such as when the node name is hard-coded in an application, use the CATALOG LDAP NODE command to make the change. The command would be similar to:

```
db2 catalog ldap node <ldap_node_name>
   as <new_alias_name>
```

To uncatalog a LDAP node, use the UNCATALOG LDAP NODE COMMAND. The command would appear similar to:

```
db2 uncatalog ldap node <ldap_node_name>
```

Deregistering the DB2 Server

Deregistration of an instance from LDAP also removes all the node, or alias, objects and the database objects referring to the instance.

Deregistration of the DB2 server on either a local or a remote machine requires the LDAP node name be specified for the server:

```
db2 deregister db2 server in ldap
   node <node_name>
```

When the DB2 server is deregistered, any LDAP node entry and LDAP database entries referring to the same instance of the DB2 server is also uncataloged.

Registration of Databases

During the creation of a database within an instance, the database is automatically registered in LDAP. Registration allows remote client connection to the database without having to catalog the database and node on the local machine. When a client attempts to connect to a database, if the database does not exist in the database directory on the local machine then the LDAP directory is searched.

If the name already exists in the LDAP directory, the database is still created on the local machine but a warning message is returned stating the naming conflict in the LDAP directory. For this reason you can manually catalog a database in the LDAP directory. The user can register databases on a remote server in LDAP by using the CATALOG LDAP DATABASE command. When registering a remote database, you specify the name of the LDAP node that represents the remote database server. You **must** register the remote database server in LDAP using the REGISTER DB2 SERVER IN LDAP command **before** registering the database.

To register a database manually in LDAP, use the CATALOG LDAP DATABASE command:

```
db2 catalog ldap database <dbname>
   at node <node_name>
   with "My LDAP database"
```

Attaching to a Remote Server

In the LDAP environment, you can attach to a remote database server using the LDAP node name on the ATTACH command:

```
db2 attach to <ldap_node_name>
```

When a client application attaches to a node or a database for the first time, since the node is not in the local node directory, DB2 searches the LDAP directory for the target node entry. If the entry is found in the LDAP directory, the database location and the protocol information of the remote server is

retrieved. Using this information, DB2 automatically catalogs a database entry and a node entry on the local machine. The next time the client application attaches to the same node or database, the information in the local database directory is used without having to search the LDAP directory.

Deregistering the Database

The database is automatically deregistered from LDAP when:

- The database is dropped.
- The owning instance is deregistered from LDAP.

The database can be manually deregistered from LDAP using:

```
db2 uncatalog ldap database <dbname>
```

Refreshing LDAP Entries in Local Database and Node Directories

LDAP information is subject to change, so it is necessary to refresh the LDAP entries in the local and node directories. The local database and node directories are used to cache the entries in LDAP.

To refresh the database entries that refer to LDAP resources, use the following command:

```
db2 refresh ldap database directory
```

To refresh the node entries on the local machine that refer to LDAP resources, use the following command:

```
db2 refresh ldap node directory
```

As part of the refresh, all the LDAP entries that are saved in the local database and node directories are removed. The next time that the application accesses the database or node, it will read the information directly from LDAP and generate a new entry in the local database or node directory.

To ensure the refresh is done in a timely way, you may want to:

- Schedule a refresh that is run periodically.
- Run the REFRESH command during system bootup.
- Use an available administration package to invoke the REFRESH command on all client machines.

Searching

DB2 searches the current LDAP directory partition. In an environment where there are multiple LDAP directory partitions or domains, you can set the search scope. For example, if the information is not found in the current partition or domain, automatic search of all other partitions or domains can be requested. On the other hand, the search scope can be restricted to search only the local machine.

The search scope is controlled through the DB2 profile registry variable, *db2ldap_search_scope*. To set the search scope value at the global level in LDAP, use the "-gl" option, which means "global in LDAP", on the *db2set* command:

```
db2set -gl db2ldap_search_scope=<value>
```

Possible values include: "local", "domain", or "global". The default value is "domain". Setting the search scope in LDAP allows the setting of the default search scope for the entire enterprise. For example, you may want to initialize the search scope to "global" after a new database is created. This allows any client machine to find a database that is defined in a particular partition or domain. Once the entry has been recorded on each machine after the first connect or attach for each client, the search scope can be changed to "local". Once changed to "local", each client will not scan any partition or domain.

Note: The DB2 profile registry variable `db2ldap_search_scope` is the only registry variable that supports setting the variable at the global level in LDAP.

Configure Host Database

You can manually configure host database information in LDAP so that each client does not need to manually catalog the database and node locally on each machine. The process follows:

1. Register the host database server in LDAP. The NODE TYPE parameter must be set to "DCS" for the REGISTER command to indicate that this is a host database server.
2. Register the host database in LDAP using the CATALOG LDAP DATABASE command. Any additional DRDA parameters can be specified by using the PARMS parameter. The database authentication type should be set to "DCS".

Setting DB2 Registry Variables at the User Level

Under the LDAP environment, the DB2 profile registry variables can be set at the user level which allows a user to customize their own DB2 environment. To set the DB2 profile registry variables at the user level, use the `-ul` option:

```
db2set -ul variable=value
```

DB2 has a caching mechanism. The DB2 profile registry variables at the user level are cached on the local machine. If the `-ul` parameter is specified, DB2 always reads from the cache for the DB2 registry variables. The cache is refreshed when:

- You update or reset a DB2 registry variable at the user level.
- You explicitly force the refresh of the registry variables at the user level by using the REFRESH command. For example:

  ```
  db2 refresh ldap profile variables
  ```

Enable LDAP Support After Installation is Complete

To enable LDAP support at some point following the completion of the installation process, use the following procedure:

- Install the LDAP support binary files. Run the setup program and select the LDAP Directory Exploitation support from `Custom` install. The setup program installs the binary files and sets the DB2 profile registry variable *db2_enable_ldap* to “YES”
- Register the current instance of the DB2 server in LDAP by using the REGISTER DB2 SERVER IN LDAP command. For example:

  ```
  db2 register db2 server in ldap protocol tcpip
  ```

- If you have databases you would like to register in LDAP, run the CATALOG LDAP DATABASE command. For example:

  ```
  db2 catalog ldap database <dbname> as <alias_dbname>
  ```

Disable LDAP Support

To disable LDAP support, use the following procedure:

- For each instance of the DB2 server, deregister the DB2 server from LDAP:

  ```
  db2 deregister db2 server in ldap node <nodename>
  ```

- Set the DB2 profile registry variable *db2_enable_ldap* to “NO”.

Security Considerations

Before accessing information in the LDAP directory, an application or user is authenticated by the LDAP server. The authentication process is called *binding* to the LDAP server.

When accessing LDAP for the first time, DB2 detects which LDAP server is being used and dynamically loads the LDAP support code using the appropriate LDAP client. If the LDAP server is IBM eNetwork Directory, the IBM LDAP client is used. The information about the server is saved in the local registry to allow the same library to be used the next time.

It is important to apply access control on the information stored in the LDAP directory to prevent anonymous users from adding, deleting, or modifying the information.

Access control is managed by the LDAP server.

Access control is inherited by default and can be applied at the container level. When a new object is created, it inherits the same security attribute as the parent object. An administration tool available for the LDAP server can be used to define access control for the container object.

By default, access control is defined as follows:

- For database and node entries in LDAP, everyone (or any anonymous user) has read access. Only the Directory Administrator and the owner or creator of the object has read / write access.
- For user profiles, the profile owner and the Directory Administrator have read / write access. One user cannot access the profile of another user if that user does not have Directory Administrator authority.

Note: The authorization check is always performed by the LDAP server and not by DB2. The LDAP authorization check is not related to DB2 authorization. An account or auth ID that has SYSADM authority may not have access to the LDAP directory.

Managing Multiple User Accounts

When running in the LDAP environment, the user account must be defined in several locations:

- In the operating system. DB2 uses the operating system account to perform DB2 authentication. The account must be defined at both the DB2 client and the DB2 server machine for DB2 server authentication. That is, before

connecting to the database: you need an account on the client machine to log on; and an account on the DB2 server machine to authenticate.

- In the LDAP directory for LDAP authentication. System management tools, such as Tivoli, or synchronization tools, are available to manage accounts in multiple places. These tools are used to ensure that any account update, like the changing of a password, is applied to all locations where the account is defined.

Extending the Directory Schema with DB2 Object Classes and Attributes

The LDAP Directory Schema defines object classes and attributes for the information stored in the LDAP directory entries. An object class consists of a set of mandatory and optional attributes. Every entry in the LDAP directory has an object class associated with it.

Before DB2 can store the information into LDAP, the Directory Schema for the LDAP server must include the object classes and attributes that DB2 uses. The process of adding new object classes and attributes to the base schema is called extending the Directory Schema.

Note: If you are using IBM SecureWay LDAP Directory v3.1, all the object classes and attributes that are required by DB2 are included in the base schema. You do not have to extend the base schema with DB2 object classes and attributes.

Extending the Directory Schema for IBM eNetwork Directory Version 2.1

When using the IBM eNetwork Directory Version 2.1, you must extend the base schema with the object classes and attributes that are used by DB2.

Use the following steps to extend the base schema for IBM eNetwork Directory Version 2.1:

1. Copy the DB2 attribute definition file, `db2.at`, and object class definition file, `db2.oc`, to the same directory that contains the system attribute and object class definition files, `slapd.at.conf` and `slapd.oc.conf`. The DB2 attribute and object class definition files can be found in the `cfg` subdirectory of the `sqllib` subdirectory. The system attribute and object class definition files are located in the `etc` subdirectory of the `%LDAPHome%` subdirectory.
2. Review the DB2 attribute and object class definition files. Comment out any object classes and attributes that have been defined in your current LDAP Directory Schema.
3. Add a line at the end of the `slapd.oc.conf` file as follows:

```
include db2.oc
```

4. Add a line at the end of the `slapd.oc.conf` file as follows:

   ```
   include db2.at
   ```

5. Restart the LDAP server.

Object Classes and Attributes Used by DB2

The following tables describe the object classes that are used by DB2:

Table 55. The eProperty Object Class

Class	**eProperty**
Description	The eProperty object class is used to specify application specific settings for user preference properties.
Required Attributes	cn
Optional Attributes	propertyType cisProperty (may contain other IBM defined attributes)
Type	Structural
OID (object identifier)	1.3.18.0.2.6.90
Special Notes	The eProperty object class is defined in the IBM Directory Schema and may contain other attributes. The attributes that are used by DB2 UDB are: 1. cn - name. The name of the eProperty object. For the DB2 Registry Variable, the object name should be “db2Env”. For CLI configuration, the object name should be “DSN - <DSN_Name>”, where `<DSN_Name>` is the CLI/ODBC Datasource Name. 2. propertyType - The type of the eProperty object. It should be set to “DB2ENV” for DB2 Registry Variable, or “DB2CLI” for CLI configuration. 3. cisProperty - a multi-value attribute, each a value contains a `keyName=keyValue` pair.

Table 56. The eApplicationSystem Object Class

Class	**eApplicationSystem**
Description	This object class describes application subsystems such as DB2 and CICS as well as systems such as Orion and GSO. An application system may span multiple computer systems or it may reside on a single computer system.
Required Attributes	

Table 56. The eApplicationSystem Object Class (continued)

Class	**eApplicationSystem**
Optional Attributes	systemName (may contain other IBM defined attributes)
Type	Structural
OID (object identifier)	1.3.18.0.2.6.8
Special Notes	The eApplicationSystem object class is defined in the IBM Directory Schema and may contain other attributes. The attribute that is used by DB2 UDB is: `systemName` which is set to "DB2"

Table 57. The DB2Node Object Class

Class	**DB2Node**
Description	This object class describes an instance of a DB2 database server.
Required Attributes	db2nodeName
Optional Attributes	db2nodeAlias db2instanceName db2Type host protocolInformation
Type	Structural
OID (object identifier)	1.3.18.0.2.6.116
Special Notes	The attributes are used by DB2 as follows: 1. db2nodeName - The name of the DB2Node object. This node name is used by client applications when connecting to the database server. 2. db2nodeAlias - Alternate node name 3. db2instanceName - The instance name of database server 4. db2Type - set to one of the following values: • SERVER - for single partition database server • MPP - for multi-partitioned database server • DCS - for host database server 5. host - the TCP/IP hostname of the machine where the server resides 6. protocolInformation - protocol specific information.

The `protocolInformation` attribute contains the communication protocol information to bind to the service. It consists of tokens that describe the network protocol supported. Each token is separated by a semicolon. There is no space between the tokens. You may specify an asterisk (*) for an optional parameter.

The tokens for TCP/IP are:

- "TCP/IP"
- server hostname or IP address
- (optional) security (NONE or SOCKS)

The tokens for APPN are:

- "APPNP"
- Network ID
- Partner LU
- Transaction Program (TP) Name
- Mode
- Security (either NONE, SAME, or PROGRAM)
- (optional) LAN adapter address
- (optional) Change password LU

The tokens for IPX/SPX are:

- "IPXSPX"
- IPX address

The tokens for NetBIOS are:

- "NETBIOS"
- Server NetBIOS workstation name

The tokens for Named Pipe are:

- "NPIPE"
- Computer name of the server
- Instance name of the server

Table 58. The DB2Database Object Class

Class	**DB2Database**
Description	This object class describes a DB2 database.
Required Attributes	db2databaseName db2nodePtr

Table 58. The DB2Database Object Class (continued)

Class	**DB2Database**
Optional Attributes	db2databaseAlias db2additionalParameters db2ARLibrary db2authenticationLocation db2gwPtr db2databaseRelease DCEPrincipalName
Type	Structural
OID (object identifier)	1.3.18.0.2.6.117
Special Notes	The attributes are described as follows: 1. db2databaseAlias - the database alias name. This is the name specified when cataloging the database. 2. db2databaseName - native database name. This is the name specified when creating the database. 3. db2nodePtr - pointer to the Node object for the database server which owns the database. This relationship allows the client application to retrieve the required protocol communication information to connect to the database server. 4. db2gwPtr - pointer to the Node object for the gateway server for non-DRDA client. In a gateway environment when there is a dedicated gateway that is configured to connect to host databases, all clients connect to the gateway before the database request is routed to the host. (To use DRDA, the user needs to install a separate product, called DB2 Connect, which allows the client to use DRDA to communicate with the host database server.) When the database protocol is set to use DRDA and DB2 Connect is not installed on the client machine, the client will connect to the gateway pointed to by the `db2gwPtr` attribute. 5. db2additionalParameters - any additional parameter used when connecting to the host database server. 6. db2ARLibrary - name of the Application Requester library 7. db2authenticationLocation - CLIENT, SERVER, DCS, or DCE 8. db2databaseRelease - database release number 9. DCEPrincipalName - when the authentication location is DCE, this attributes contains the DCE Principal Name.

The following table describes the attributes that are used by DB2:

Table 59. The Attribute Specifications

Attribute Name	Syntax	Maximum Length	Multi-Valued	OID
cn	Case Ignored String	256	Multi-valued	2.5.4.3
propertyType	Case Ignored String	64	Multi-valued	1.3.18.0.2.4.320
cisProperty	Case Ignored String	250000	Multi-valued	1.3.18.0.2.4.309
systemName	Case Ignored String	256	Single-valued	1.3.18.0.2.4.329
db2nodeName	Case Ignored String	1024	Single-valued	1.3.18.0.2.4.419
db2nodeAlias	Case Ignored String	1024	Multi-valued	1.3.18.0.2.4.420
db2instanceName	Case Ignored String	256	Single-valued	1.3.18.0.2.4.428
db2Type	Case Ignored String	64	Single-valued	1.3.18.0.2.4.418
host	Case Ignored String	256	Multi-valued	1.3.18.0.2.4.486
protocolInformation	binary	5000	Multi-valued	2.5.4.48
db2databaseName	Case Ignored String	1024	Single-valued	1.3.18.0.2.4.421
db2databaseAlias	Case Ignored String	1024	Multi-valued	1.3.18.0.2.4.422
db2nodePtr	Distinguished Name	1000	Single-valued	1.3.18.0.2.4.423
db2gwPtr	Distinguished Name	1000	Single-valued	1.3.18.0.2.4.424
db2additionalParameters	Case Ignored String	1024	Single-valued	1.3.18.0.2.4.426
db2ARLibrary	Case Ignored String	256	Single-valued	1.3.18.0.2.4.427
db2authenticationLocation	Case Ignored String	64	Single-valued	1.3.18.0.2.4.425
db2databaseRelease	Case Ignored String	64	Single-valued	1.3.18.0.2.4.429

Table 59. The Attribute Specifications (continued)

Attribute Name	Syntax	Maximum Length	Multi-Valued	OID
DCEPrincipalName	Case Ignored String	2048	Single-valued	1.3.18.0.2.4.443

Appendix O. Extending the Control Center

In Version 6, you can extend the DB2 Universal Database Control Center by using the new *plug-in* architecture to provide additional function.

The concept of the *plug-in* architecture is to provide the ability to add items for a given object in the Control Center popup menu, and add new buttons to the tool bar. A set of Java interfaces, which you must implement, is shipped along with the tools. These interfaces are used to communicate to the Control Center what additional actions to include.

Performance Considerations

The plug-in extensions (db2plug.zip) are loaded at the startup time of the Control Center tools. This may increase the startup time of the tools, depending on the size of the ZIP file; however, we expect that the plug-in ZIP file will be small for most users and the impact should be minimal.

Packaging Considerations

You must ZIP the extension class files according to the rules of a Java archive file. To run the Control Center tools as an application, the ZIP file (db2plug.zip) must be in the classpath. To run the Control Center tools as an applet, the ZIP file must be located where the <codebase> tag points to in the Control Center html file.

The ZIP file should be built will no compression and maintain the relative path positions of all the class files (`zip -r0 db2plug.zip *.class`).

Interface Descriptions

The following interfaces are shipped:

- CCExtension
- CCObject
- CCMenuAction
- CCToolbarAction.

The interfaces are described in the next sections, followed by an example.

CCExtension

The CCExtension interface allows you to extend the Control Center user interface by adding new toolbar buttons, new menu items, and overriding existing menu actions.

The external interface is defined as follows:

```
public interface CCExtension
{
   /**
    * Get an array of CCObject subclass objects which define
    * a list of objects to be inserted or overridden in the
    * Control Center
    * @return CCObject[] CCObject subclass objects array
    */
   public CCObject[] getObjects();

   /**
    * Get an array of CCToolbarAction subclass objects which represent
    * a list of buttons to be added to the Control Center
    * main toolbar.
    * @return CCToolbarAction[] CCToolbarAction subclass objects array
    */
   public CCToolbarAction[] getToolbarActions();
}
```

To use CCExtension, create a Java class which imports the ″com.ibm.db2.tools.cc.navigator″ package and implements this interface. The new class must provide the implementation of the getObjects() and getToolbarActions() methods.

The getObjects() method returns an array of CCObject which defines the existing objects which the user would like to add new menu actions or remove a predefined set of menu actions.

The getToolbarActions() method returns an array of CCToolbarAction which will be added to the Control Center main toolbar.

A single CCExtension subclass file or multiple CCExtension subclass files can be used to define the Control Center extensions. For the Control Center to use these extensions, use the following setup procedure:

1. Create a ″db2plug.zip″ file which contains all the CCExtension subclass files. The files should not be compressed. For example, if the CCExtension files are in the plugin package and they are located in the plugin directory,

   ```
   zip -r0 db2plug.zip plugin\*.class
   ```

   This command will put all the plugin package class files into the db2plug.zip file and preserve their relative path information.

2. To run the Control Center as an applet, put the db2plug.zip file in where the <codebase> tag points to in the Control Center html file. To run the Control Center as an application, put the db2plug.zip in a directory pointed to by the CLASSPATH environment variable.

For browsers that support multiple archives, just add ″db2plug.zip″ to the archive list of the Control Center html page. Otherwise, all the CCExtension, CCObject, CCToolbarAction, and CCMenuAction subclass files will have to be in their relative directories depending on which package they belong to.

CCObject

The CCObject interface allows you to change the behavior of the menu actions of an existing object.

The external interface is defined as follows:

```
public interface CCObject
{
   /**
    * The following static constants defines a list of object type
    * available to be added to the Control Center tree.
    */
   public static final int UDB_SYSTEMS_FOLDER                          = 0;
   public static final int UDB_SYSTEM                                  = 1;
   public static final int UDB_INSTANCES_FOLDER                        = 2;
   public static final int UDB_INSTANCE                                = 3;
   public static final int UDB_DATABASES_FOLDER                        = 4;
   public static final int UDB_DATABASE                                = 5;
   public static final int UDB_TABLES_FOLDER                           = 6;
   public static final int UDB_TABLE                                   = 7;
   public static final int UDB_TABLESPACES_FOLDER                      = 8;
   public static final int UDB_TABLESPACE                              = 9;
   public static final int UDB_VIEWS_FOLDER                            = 10;
   public static final int UDB_VIEW                                    = 11;
   public static final int UDB_ALIASES_FOLDER                          = 12;
   public static final int UDB_ALIAS                                   = 13;
   public static final int UDB_TRIGGERS_FOLDER                         = 14;
   public static final int UDB_TRIGGER                                 = 15;
   public static final int UDB_SCHEMAS_FOLDER                          = 16;
   public static final int UDB_SCHEMA                                  = 17;
   public static final int UDB_INDEXES_FOLDER                          = 18;
   public static final int UDB_INDEX                                   = 19;
   public static final int UDB_CONNECTIONS_FOLDER                      = 20;
   public static final int UDB_CONNECTION                              = 21;
   public static final int UDB_REPLICATION_SOURCES_FOLDER              = 22;
   public static final int UDB_REPLICATION_SOURCE                      = 23;
   public static final int UDB_REPLICATION_SUBSCRIPTIONS_FOLDER        = 24;
   public static final int UDB_REPLICATION_SUBSCRIPTION                = 25;
   public static final int UDB_BUFFERPOOLS_FOLDER                      = 26;
   public static final int UDB_BUFFERPOOL                              = 27;
   public static final int UDB_APPLICATION_OBJECTS_FOLDER              = 28;
   public static final int UDB_USER_DEFINED_DISTINCT_DATATYPES_FOLDER  = 29;
   public static final int UDB_USER_DEFINED_DISTINCT_DATATYPE          = 30;
```

```
public static final int UDB_USER_DEFINED_DISTINCT_FUNCTIONS_FOLDER  = 31;
public static final int UDB_USER_DEFINED_DISTINCT_FUNCTION          = 32;
public static final int UDB_PACKAGES_FOLDER                         = 33;
public static final int UDB_PACKAGE                                 = 34;
public static final int UDB_STORE_PROCEDURES_FOLDER                 = 35;
public static final int UDB_STORE_PROCEDURE                         = 36;
public static final int UDB_USER_AND_GROUP_OBJECTS_FOLDER           = 37;
public static final int UDB_DB_USERS_FOLDER                         = 38;
public static final int UDB_DB_USER                                 = 39;
public static final int UDB_DB_GROUPS_FOLDER                        = 40;
public static final int UDB_DB_GROUP                                = 41;
public static final int UDB_DRDA_TABLE                              = 42;

public static final int S390_SUBSYSTEMS_FOLDER                      = 43;
public static final int S390_SUBSYSTEM                              = 44;
public static final int S390_BUFFERPOOLS_FOLDER                     = 45;
public static final int S390_BUFFERPOOL                             = 46;
public static final int S390_VIEWS_FOLDER                           = 47;
public static final int S390_VIEW                                   = 48;
public static final int S390_DATABASES_FOLDER                       = 49;
public static final int S390_DATABASE                               = 50;
public static final int S390_TABLESPACES_FOLDER                     = 51;
public static final int S390_TABLESPACE                             = 52;
public static final int S390_TABLES_FOLDER                          = 53;
public static final int S390_TABLE                                  = 54;
public static final int S390_INDEXS_FOLDER                          = 55;
public static final int S390_INDEX                                  = 56;
public static final int S390_STORAGE_GROUPS_FOLDER                  = 57;
public static final int S390_STORAGE_GROUP                          = 58;
public static final int S390_ALIASES_FOLDER                         = 59;
public static final int S390_ALIAS                                  = 60;
public static final int S390_SYNONYMS_FOLDER                        = 61;
public static final int S390_SYNONYM                                = 62;
public static final int S390_APPLICATION_OBJECTS_FOLDER             = 63;
public static final int S390_COLLECTIONS_FOLDER                     = 64;
public static final int S390_COLLECTION                             = 65;
public static final int S390_PACKAGES_FOLDER                        = 66;
public static final int S390_PACKAGE                                = 67;
public static final int S390_PLANS_FOLDER                           = 68;
public static final int S390_PLAN                                   = 69;
public static final int S390_PROCEDURES_FOLDER                      = 70;
public static final int S390_PROCEDURE                              = 71;
public static final int S390_DB_USERS_FOLDER                        = 72;
public static final int S390_DB_USER                                = 73;
public static final int S390_LOCATIONS_FOLDER                       = 74;
public static final int S390_LOCATION                               = 75;
public static final int S390_DISTINCT_TYPES_FOLDER                  = 76;
public static final int S390_DISTINCT_TYPE                          = 77;
public static final int S390_USER_DEFINED_FUNCTIONS_FOLDER          = 78;
public static final int S390_USER_DEFINED_FUNCTION                  = 79;
public static final int S390_TRIGGERS_FOLDER                        = 80;
public static final int S390_TRIGGER                                = 81;
public static final int S390_SCHEMAS_FOLDER                         = 82;
public static final int S390_SCHEMA                                 = 83;
public static final int S390_CATALOG_TABLES_FOLDER                  = 84;
```

```
public static final int S390_CATALOG_TABLE                          = 85;

public static final int DCS_GATEWAY_CONNECTIONS_FOLDER              = 86;
public static final int DCS_GATEWAY_CONNECTION                      = 87;

/**
 * Total number of object types
 */
public static final int NUM_OBJECT_TYPES                            = 88;

/**
 * Get the name of these object
 * The function returns the name of this object. This name
 * can be of three types:
 * (1) Fully qualified name
 *     Syntax: xxxxx-yyyyy-zzzzz
 *             where xxxxx-yyyyy is the fully quality name of the
 *             parent object and zzzzz is the name of the new object.
 *     Note: Parent and child object name is separated by '-' character.
 *     If a schema name is required to identify object, the fully
 *     qualified name is represented by xxxxx-yyyyy-wwwww.zzzzz
 *     where wwwww is the schema name.
 *     Only the behavior of the object that matches this fully
 *     qualified name will be affected.
 * (2) Parent fully qualified name
 *     Syntax: xxxxx-yyyyy
 *             where xxxxx-yyyyy is the fully qualified name of the
 *             parent object.
 *     When the object type is folder (ie. DATABASES_FOLDER), the
 *     getName() should only return the fully qualified name of the
 *     folder's parent.
 *     Only the behavior of the object that match this name
 *     and the specific type return by the getType() function will be
 *     affected.
 * (3) null
 *     Syntax: null
 *     If null is return, the CCActions returns by the getActions()
 *     call will be applied to all objects of type returns by the
 *     getType() call.
 * @return String object name
 */
public String getName();

/**
 * Get the type of this object
 * @return int return one of the static type constants defined
 * in this interface
 */
public int getType();

/**
 * Get the CCMenu Action array which defines the list of menu actions
 * to be created for the selected object
 * return CCMenuAction[] CCMenuAction array
 */
```

```
   public CCMenuAction[] getMenuActions();

   /**
    * Check if this object is editable.
    * If not, the Alter related menu items will be removed from
    * the object's popup menu return boolean If false, the Alter
    * menu item will be removed from the object's popup menu
    */
   public boolean isEditable();

   /**
    * Check if this object is configurable.
    * If not, the configuration related menu items will be
    * removed from the object's popup menu return boolean If
    * false, the Configuration related menu item will be removed
    * from the object's popup menu
    */
   public boolean isConfigurable();
}
```

Note: At this time, the last two methods in CCObject: isEditable() and isConfigurable() should always return **true**.

CCMenuAction

The CCMenuAction interface allows you to define a new action to be used by a Control Center object.

The external interface is defined as follows:

```
public interface CCMenuAction
{
   /**
    * Get the name of this action
    * @return String Name text on the menu item
    */
   public String getMenuText();

   /**
    * Invoked when an action occurs. Use the getActionCommand()
    * method of the ActionEvent to get the fully qualified name of
    * the invoked Control Center object.
    * @param e Action event
    */
   public void actionPerformed(ActionEvent e);
}
```

CCToolBarAction

The CCToolbarAction interface allows you to define a new action on the Control Center toolbar.

The external interface is defined as follows:

```
public interface CCToolbarAction
{
   /**
    * Get the name of this action
    * @return String Name text on the menu item, or toolbar
    * button hover help
    */
   public String getHoverHelpText();

   /**
    * Get the icon for the toolbar button
    * Any toolbar CCAction should implement this function and return
    * a valid ImageIcon object. Otherwise, the button will have no icon.
    * @return ImageIcon Icon to be displayed
    */
   public ImageIcon getIcon();

   /**
    * Invoked when an action occurs.
    * @param e Action event
    */
   public void actionPerformed(ActionEvent e);
}
```

Usage Scenario

The code in the following example will:

1. Update the actions of the SAMPLE Database (see “MySample.java” on page 852)
2. Update the actions of all Database objects (see “MyDatabaseActions.java” on page 853)
3. Add a new instance object (see “MyInstance.java” on page 854)
4. Update the actions of the DB2 instance (see “MyDB2.java” on page 854)
5. Update the actions of the Databases folder (see “MyDatabases.java” on page 855)
6. Update the actions of the SYSIBM.SYSPLAN table (see “MySYSPLAN.java” on page 856)
7. Add a new table object (see “MyTable.java” on page 856)
8. Update the actions of the DB_User object under the Application object (see “MyDBUser.java” on page 857)
9. Add a button to the Control Center toolbar (see “MyToolbarAction.java” on page 858).

The main extension file is MyExtension.java. All the class files are stored in the plugin directory and are zipped up by the command:

```
zip -r0 db2plug.zip plugin
```

The output db2plug.zip is then placed in the CLASSPATH or in the codebase directory depending whether the Control Center is running as an application or an applet.

MyExtension.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyExtension implements CCExtension
{
   public CCObject[] getObjects()
   {
      CCObject[] objs = new CCObject[10];
      objs[0] = new MySample();
      objs[1] = new MyDatabaseActions();
      objs[2] = new MyInstance();
      objs[3] = new MyDB2();
      objs[4] = new MyDatabases();
      objs[5] = new MySYSPLAN();
      objs[6] = new MyTable();
      objs[7] = new MyDBUser();
      return objs;
   }

   public CCAction[] getActions()
   {
      CCAction[] actions = new CCAction[1];
      actions[0] = new MyToolbarAction();
      return actions;
   }
}
```

MySample.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MySample implements CCObject
{
   public String getName()
   {
      return "LOCAL - DB2 - SAMPLE";
   }

   public int getType()
   {
      return DATABASE;
   }

   public javax.swing.ImageIcon getIcon()
   {
      return null;
   }
```

```
   public boolean isNew()
   {
      return false;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[2];
      acts[0] = new MyAlterAction();
      acts[1] = new MyAction();
      return acts;
   }

}
```

MyDatabaseActions.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyDatabaseActions implements CCObject
{
   public String getName()
   {
      return null;
   }

   public int getType()
   {
      return DATABASE;
   }

   public javax.swing.ImageIcon getIcon()
   {
      return null;
   }

   public boolean isNew()
   {
      return false;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[2];
      acts[0] = new MyDropAction();
      acts[1] = new MyAction();
      return acts;
   }

}
```

MyInstance.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyInstance implements CCObject
{
   public String getName()
   {
      return "LOCAL - MyInstance";
   }

   public int getType()
   {
      return INSTANCE;
   }

   public javax.swing.ImageIcon getIcon()
   {
      return null;
   }

   public boolean isNew()
   {
      return true;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[2];
      acts[0] = new MyAlterAction();
      acts[1] = new MyAction();
      return null;
   }

}
```

MyDB2.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyDB2 implements CCObject
{
   public String getName()
   {
      return "LOCAL - DB2";
   }

   public int getType()
   {
      return INSTANCE;
   }

   public javax.swing.ImageIcon getIcon()
   {
```

```
      return null;
   }

   public boolean isNew()
   {
      return false;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[3];
      acts[0] = new MyAlterAction();
      acts[1] = new MyAction();
      acts[2] = new MyCascadeAction();
      return acts;
   }
}
```

MyDatabases.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyDatabases implements CCObject
{
   public String getName()
   {
      return "LOCAL - DB2 - Databases";
   }

   public int getType()
   {
      return DATABASE;
   }

   public javax.swing.ImageIcon getIcon()
   {
      return null;
   }

   public boolean isNew()
   {
      return false;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[1];
      acts[0] = new MyCreateAction();
      return acts;
   }

}
```

MySYSPLAN.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MySYSPLAN implements CCObject
{
   public String getName()
   {
      return "LOCAL - DB2 - SAMPLE - SYSIBM - SYSPLAN";
   }

   public int getType()
   {
      return TABLE;
   }

   public javax.swing.ImageIcon getIcon()
   {
      return null;
   }

   public boolean isNew()
   {
      return false;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[2];
      acts[0] = new MyAlterAction();
      acts[1] = new MyAction();
      return acts;
   }

}
```

MyTable.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyTable implements CCObject
{
   public String getName()
   {
      return "LOCAL - DB2 - SAMPLE - SYSIBM - MyTable";
   }

   public int getType()
   {
      return TABLE;
   }

   public javax.swing.ImageIcon getIcon()
   {
```

```
      return null;
   }

   public boolean isNew()
   {
      return true;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[2];
      acts[0] = new MyAlterAction();
      acts[1] = new MyAction();
      return acts;
   }

}
```

MyDBUser.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyDBUser implements CCObject
{
   public String getName()
   {
      return "LOCAL - DB2 - TEST-DB Users";
   }

   public int getType()
   {
      return DB_USER;
   }

   public javax.swing.ImageIcon getIcon()
   {
      return null;
   }

   public boolean isNew()
   {
      return false;
   }

   public CCAction[] getActions()
   {
      CCAction[] acts = new CCAction[2];
      acts[0] = new MyAlterAction();
      acts[1] = new MyAction();
      return acts;
   }

}
```

MyToolbarAction.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;
import javax.swing.*;

public class MyToolbarAction extends CCAction
{
   public MyToolbarAction()
   {
      super("MyToolbarAction");
   }

   public ImageIcon getIcon()
   {
      return <Your icon>;
   }

   public boolean actionPerformed(String objectName)
   {
      System.out.println( "My action performed, object name = " +
                          objectName );
      return true;
   }
}
```

MyAlterAction.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyAlterAction extends CCAction
{
   public MyAlterAction()
   {
      super(0);
   }

   public boolean actionPerformed(String objectName)
   {
      System.out.println( "My alter action performed, object name = " +
                          objectName );
      return true;
   }
}
```

MyAction.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyAction extends CCAction
{
   public MyAction()
   {
      super("MyAction");
   }
```

```
   public boolean actionPerformed(String objectName)
   {
      System.out.println( "My action performed, object name = " +
                          objectName );
      return true;
   }
}
```

MyDropAction.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyDropAction extends CCAction
{
   public MyDropAction()
   {
      super(1);
   }

   public boolean actionPerformed(String objectName)
   {
      System.out.println( "My drop action performed, object name = " +
                          objectName );
      return true;
   }
}
```

MyCascadeAction.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyCascadeAction extends CCAction
{
   public MyCascadeAction()
   {
      super(11,2);
   }

   public boolean actionPerformed(String objectName)
   {
      System.out.println( "My cascade action performed, object name = " +
                          objectName );
      return true;
   }
}
```

MyCreateAction.java

```
package plugin;
import com.ibm.db2.tools.cc.navigator.*;

public class MyCreateAction extends CCAction
{
   public MyCreateAction()
```

```
    {
       super(0);
    }

    public boolean actionPerformed(String objectName)
    {
       System.out.println( "My create action performed, object name = " +
                           objectName );
       return true;
    }
}
```

Appendix P. 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, North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (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
Office of the Lab Director
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
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
LAN Distance
MVS/ESA
MVS/XA
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
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. in the United States, other countries, or both.

HP-UX is a trademark of Hewlett-Packard.

Java, HotJava, Solaris, Solstice, and Sun are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, Visual Basic, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

PC Direct is a trademark of Ziff Communications Company in the United States, other countries, or both and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark in the United States, other countries or both and is 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 accessing the following page:

`http://www.ibm.com/support/`

then performing a search using the keyword “handbook”.

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

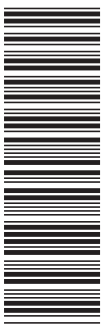
All DB2 products are supported through these forums.

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

Part Number: CT6DANA

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

CT6DANA



SC09-2839-00