IBM® DB2® Universal Database

Administration Guide: Planning

Version 7

SC09-2946-01

IBM® DB2® Universal Database

Administration Guide: Planning

Version 7

SC09-2946-01

Before using this information and the product it supports, be sure to read the general information under “Appendix F. Notices” on page 285.

Contents

About This Book

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

- Information about database design (found in *Administration Guide: Planning*)
- Information about implementing and managing databases (found in *Administration Guide: Implementation*)
- Information about configuring and tuning 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 Line 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 line processor, see the *Command Reference*.
- 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*.
- 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 set up the replication of data between systems. Further, the Control Center allows you to execute DB2 utility functions through a graphical user interface. There are different methods to invoke the Control Center depending on your platform. For example, use the `db2cc` command on a command line, (on OS/2) select the Control Center icon from the DB2 folder, or use start panels on Windows platforms. For introductory help, select **Getting started** from the **Help** pull-down of the Control Center window. The **Visual Explain** and **Performance Monitor** tools are invoked from the Control Center.

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

- The `Script Center` to store small applications called scripts. These scripts may contain SQL statements, 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 that are to run unattended.
- The `Data Warehouse Center` to manage warehouse objects.

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

This book contains information about the following major topics:

The World of DB2 Universal Database

- Chapter 1. Administering DB2 Universal Database, presents an introduction to, and overview of, DB2 Universal Database.

Database Concepts

- Chapter 2. Basic Relational Database Concepts, presents an overview of database objects, including storage objects and system objects.
- Chapter 3. Federated Systems, discusses federated systems, which are database management systems (DBMSs) that support applications and users submitting SQL statements referencing two or more DBMSs or databases in a single statement.
- Chapter 4. Parallel Database Systems, provides an introduction to the types of parallelism available with DB2.
- Chapter 5. About Data Warehousing, provides an overview of data warehousing and data warehousing tasks.
- Chapter 6. About Spatial Extender, introduces Spatial Extender by explaining its purpose and discussing the data that it processes.

Database Design

- Chapter 7. Logical Database Design, discusses the concepts and guidelines for logical database design.
- Chapter 8. Physical Database Design, discusses the guidelines for physical database design, including considerations related to data storage.

- Chapter 9. Designing Distributed Databases, discusses how you can access multiple databases in a single transaction.
- Chapter 10. Designing for Transaction Managers, discusses how you can use your databases in a distributed transaction processing environment, such as CICS.
- Chapter 11. Introducing High Availability and Failover Support, presents an overview of the high availability failover support that is provided by DB2.

Appendixes

- Appendix A. Naming Rules, presents the rules to follow when naming databases and objects.
- Appendix B. Planning Database Migration, provides information about migrating databases to Version 7.
- Appendix C. Incompatibilities Between Releases, presents the incompatibilities introduced from release to release up to, and including, Version 7.
- Appendix D. National Language Support (NLS), introduces DB2 National Language Support, including information about countries, languages, and code pages.
- Appendix E. Using the DB2 Library, provides information about the structure of the DB2 library, including wizards, online help, messages, and books.

A Brief Overview of the Other Volumes of the Administration Guide

Administration Guide: Implementation

The *Administration Guide: Implementation* is concerned with the implementation of your database design. The specific chapters and appendixes in that volume are briefly described here:

Administering Using the Control Center

- ″Administering DB2 Using GUI Tools″ describes the graphical user interface (GUI) tools used to administer the database.

Implementing Your Design

- ″Before Creating a Database″ describes the prerequisites before you create a database.
- ″Creating a Database″ describes those tasks associated with the creation of a database and related database objects.
- ″Altering a Database″ discusses what must be done before altering a database and those tasks associated with the modifying or dropping of a database or related database objects.

Database Security

- ″Controlling Database Access″ describes how you can control access to your database's resources.
- ″Auditing DB2 Activities″ describes how you can detect and monitor unwanted or unanticipated access to data.

Moving Data

- ″Utilities for Moving Data″ is a one-page introduction to the different ways to move data and to direct you to the *Data Movement Utilities Guide and Reference* book.

Recovery

- ″Recovering a Database″ is a one-page introduction to the concepts of database backup, restore and rollforward. More extensive information can be found in the *Data Recovery and High Availability Guide and Reference*.

Appendixes

- ″Using Distributed Computing Environment (DCE) Directory Services″ discusses information about how you can use DCE Directory Services.
- ″User Exit for Database Recovery″ discusses how user exit programs can be used with database log files, and describes some sample user exit programs.
- ″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.
- ″How DB2 for Windows NT Works with Windows NT Security″ describes how DB2 works with Windows NT security.
- ″Using the Windows NT Performance Monitor″ provides information about registering DB2 with the Windows NT Performance Monitor, and using the performance information.
- ″Working with Windows NT or Windows 2000 Database Partition Servers″ provides information about the utilities available to work with database partition servers on Windows NT or Windows 2000.
- ″Configuring Multiple Logical Nodes″ describes how to configure multiple logical nodes in a partitioned database environment.
- ″High Speed Inter-node Communications″ describes how to enable Virtual Interface Architecture for use with DB2 Universal Database.
- ″Lightweight Directory Access Protocol (LDAP) Directory Services″ provides information about how you can use LDAP Directory Services.
- ″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.

Administration Guide: Performance

The *Administration Guide: Performance* is concerned with performance issues; that is, those topics and issues concerned with establishing, testing, and improving the performance of your application, and that of the DB2 Universal Database product itself. 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.
- ″Architecture and Processing Overview″ introduces underlying DB2 Universal Database architecture and processes.

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″ describes an overview of how the database manager uses memory and other considerations that affect run-time performance.
- ″Using the Governor″ describes an introduction to the use of a governor to control some aspects of database management.
- ″Scaling Your Configuration″ describes 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″ presents 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″ describes profile registry values and environment variables.
- ″Explain Tables and Definitions″ describes the tables used by the DB2 Explain facility and how to create those tables.
- ″SQL Explain Tools″ describes how to use the DB2 explain tools: db2expln and dynexpln.
- ″db2exfmt — Explain Table Format Tool″ describes how to use the DB2 explain tool to format the explain table data.

Part 1. The World of DB2 Universal Database

Chapter 1. Administering DB2 Universal Database

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.

DB2 also supports many different levels of complexity in database environments, and there are considerations and tasks specific to each environment. These are discussed in detail in both the *Administration Guide* and other books in the DB2 library (see "Appendix E. Using the DB2 Library" on page 267). In some cases, entire sections of these books are only appropriate for a specific environment. After reading the preface to this book ("About This Book"), you will understand which chapters in this and the other volumes of the *Administration Guide* (the *Administration Guide: Implementation*, and the *Administration Guide: Performance*) are appropriate for your business needs.

If you are new to relational database management systems (RDBMSs), or to DB2, you will find the section entitled "Basic Relational Database Concepts" helpful. If you are familiar with these concepts, or do not need to review them, you can skip this section and move directly to the sections detailing more advanced topics, such as:

- Federated systems. This section discusses database management systems (DBMSs) that support applications and users submitting SQL statements referencing two or more DBMSs or databases in a single statement.
- Parallel database systems. This section provides an introduction to the types of parallelism available with DB2. Components of a task, such as a database query, can be run in parallel to dramatically enhance performance.
- Distributed transaction processing. This section discusses how you can access multiple databases in a single transaction, and how you can use your databases in a distributed transaction processing environment.

DB2 can address your most specialized data management needs, such as:

- *Replication*, which allows you to copy data on a regular basis to multiple remote databases. If you need updates from a master database to be copied automatically to other databases, you can use the replication features of DB2 to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied. If you want to use the replication features of DB2, refer to the *Replication Guide and Reference*. It introduces the concepts of DB2 data replication, and it describes how to plan, configure, and administer a replication environment.

- *Data warehousing*, in which you can create stores of "informational data", or data that is extracted from operational data and then transformed for end-user decision making. For example, a data warehousing tool might copy all the sales data from the operational database, perform calculations to summarize the data, and write the summarized data to a target in a separate database. You can query the separate database (the *warehouse*) without impacting the operational databases. For detailed information about data warehousing, refer to the *Data Warehouse Center Administration Guide*.
- A *geographic information system* (GIS), which can be created through Spatial Extender. A GIS is a complex of objects, data, and applications that allows you to generate and analyze spatial information about geographic features. In Spatial Extender, a geographic feature can be represented by a row in a table or view, or by a portion of such a row. For detailed information about using Spatial Extender, refer to the *Spatial Extender User's Guide and Reference*.

The *Administration Guide: Planning* also covers database design, including logical database design and physical database design considerations for DB2. Other planning issues, such as planning database migration, identifying incompatibilities that might impact your applications (an *incompatibility* is a part of DB2 Universal Database that works differently than it did in a previous release of DB2; if used in an existing application, it will produce an unexpected result, necessitate a change to the application, or reduce performance), and exploiting national language support (NLS), are also discussed.

The *Administration Guide: Implementation* covers the details of implementing your database design. Topics include creating and altering a database, database security, and administering DB2 using the Control Center, a DB2 graphical user interface.

The *Administration Guide: Performance* covers topics and issues concerned with establishing, testing, and improving the performance of your application and of DB2 itself.

Part 2. Database Concepts

Chapter 2. Basic Relational Database Concepts

This section covers the following topics:

Overview of Database Objects

This section provides an overview of the following key database objects:

- Instances
- Databases
- Nodegroups
- Tables
- Views
- Indexes
- Schemas
- System catalog tables

Figure 1 on page 8 illustrates the relationship among some of these objects. It also shows that tables, indexes, and long data are stored in table spaces.

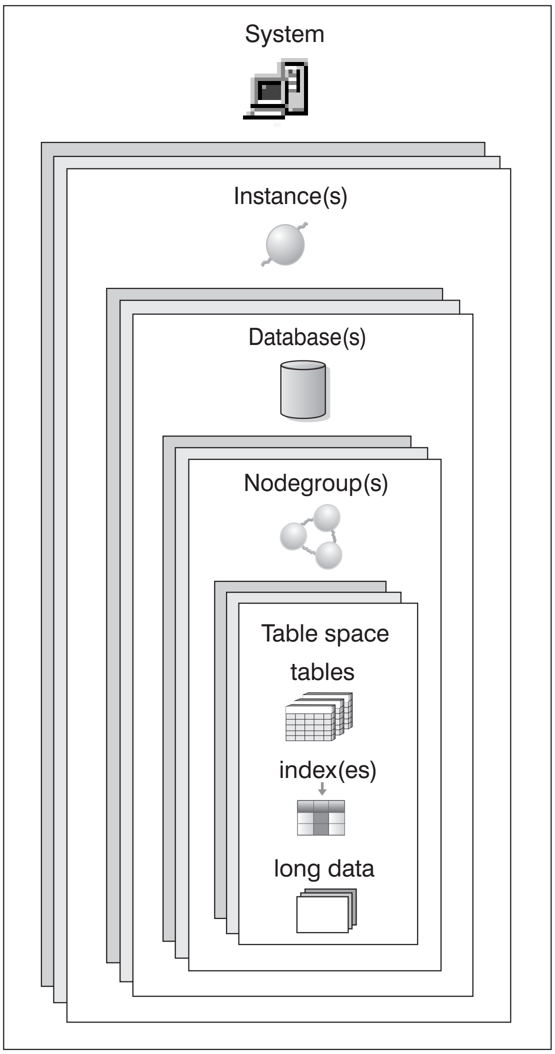


Figure 1. Relationships Among Some Database Objects

Instances

An *instance* (sometimes called a *database manager*) 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 (see "Chapter 4. Parallel Database Systems" on page 33). 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 (system).

Databases

A *relational database* presents data as a collection of tables. A table consists of a defined number of columns and any number of rows. Each database includes a set of system catalog tables that describe the logical and physical structure of the data, a configuration file containing the parameter values allocated for the database, and a recovery log with ongoing transactions and archivable transactions.

Nodegroups

A *nodegroup* is a set of one or more database partitions. When you want to create tables for the database, you first create the nodegroup where the table spaces will be stored, then you create the table space where the tables will be stored. See “Nodegroups and Data Partitioning” on page 33 for more information about nodegroups. See “Chapter 4. Parallel Database Systems” on page 33 for the definition of a database partition. See “Table Spaces” on page 12 for more information about table spaces.

Tables

A relational database presents data as a collection of tables. A *table* consists of data logically arranged in columns and rows. All database and table data is assigned to table spaces. See “Table Spaces” on page 12 for more information about table spaces. 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 through Structured Query Language (SQL, see the *SQL Reference*), a standardized language for defining and manipulating data in a relational database. A *query* is used in applications or by users to retrieve data from a database. The query uses SQL to create a statement in the form of

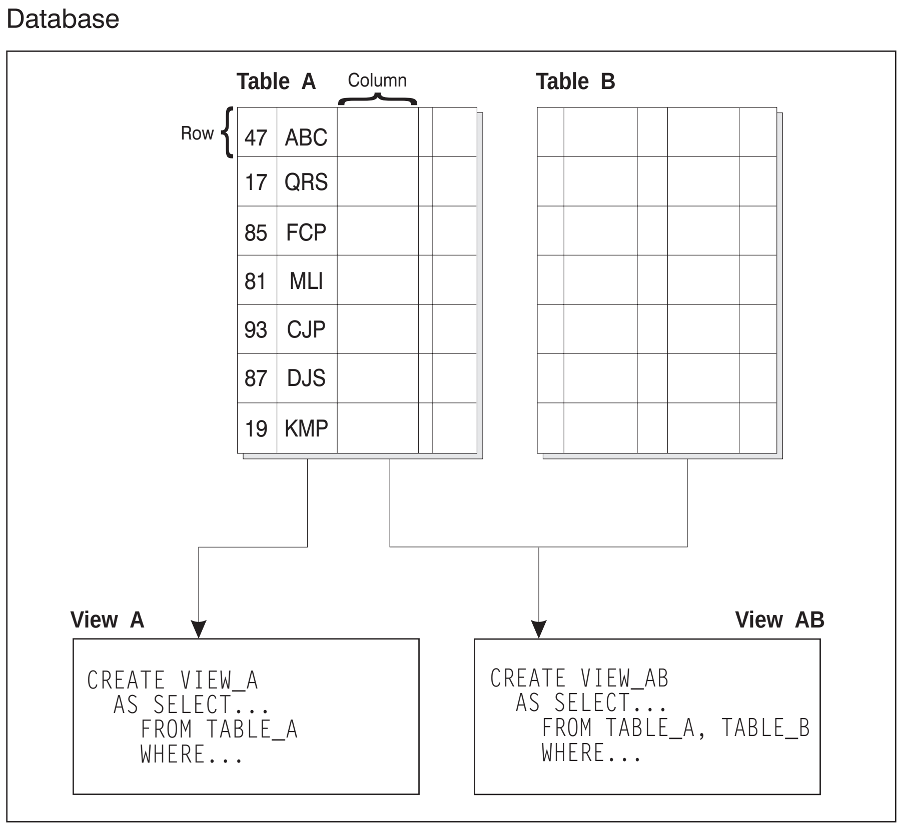
```
SELECT <data_name> FROM <table_name>
```

Views

A *view* is an efficient way of representing data without needing to maintain it. A view is not an actual table and requires no permanent storage. A ″virtual table″ is created and used.

A view can include all or some of the columns or rows contained in the tables on which it is based. For example, you can join a department table and an employee table in a view, so that you can list all employees in a particular department.

Figure 2 on page 10 shows the relationship between tables and views.



Database
Table A
Column
Table B
Row
47 ABC
17 QRS
85 FCP
81 MLI
93 CJP
87 DJS
19 KMP
View A
View AB
CREATE VIEW_A AS SELECT... FROM TABLE_A WHERE...
CREATE VIEW_AB AS SELECT... FROM TABLE_A, TABLE_B WHERE...

Figure 2. Relationship Between Tables and Views

Indexes

An *index* is a set of keys, each pointing to rows in a table. For example, table A in Figure 3 on page 11 has an index based on the employee numbers in the table. This key value provides a pointer to the rows in the table: employee number 19 points to employee KMP. An index allows more efficient access to rows in a table by creating a direct path to the data through pointers.

The SQL *optimizer* automatically chooses the most efficient way to access data in tables. The optimizer takes indexes into consideration when determining the fastest access path to data.

Unique indexes can be created to ensure uniqueness of the index key. An *index key* is a column or an ordered collection of columns on which an index is defined. Using a unique index will ensure that the value of each index key in the indexed column or columns is unique. “Business Rules for Data” on page 19 describes keys and indexes in more detail.

Figure 3 shows the relationship between an index and a table.

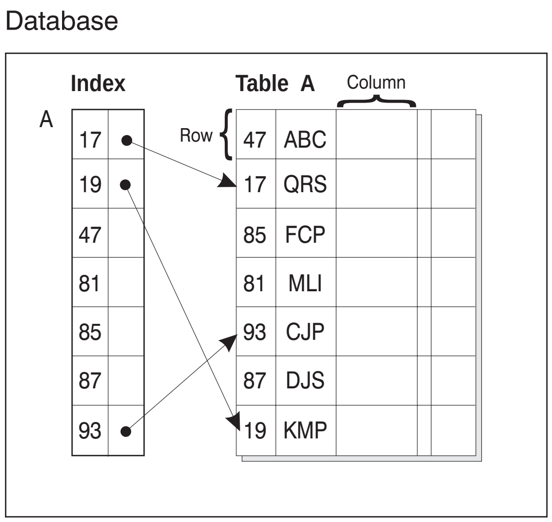


Figure 3. Relationship Between an Index and a Table

Schemas

A *schema* is an identifier, such as a user ID, that helps group tables and other database objects. A schema can be owned by an individual, and the owner can control access to the data and the objects within it.

A schema is also an object in the database. It may be created automatically when the first object in a schema is created. Such an object can be anything that can be qualified by a schema name, such as a table, index, view, package, distinct type, function, or trigger. You must have IMPLICIT_SCHEMA authority if the schema is to be created automatically, or you can create the schema explicitly.

A schema name is used as the first part of a two-part object name. When an object is created, you can assign it to a specific schema. If you do not specify a schema, it is assigned to the default schema, which is usually the user ID of the person who created the object. The second part of the name is the name of the object. For example, a user named Smith might have a table named SMITH.PAYROLL.

System Catalog Tables

Each database includes a set of *system catalog tables*, which describe the logical and physical structure of the data. DB2 creates and maintains an extensive set of system catalog tables for each database. These tables contain information about the definitions of database objects such as user tables, views, and indexes, as well as security information about the authority that users have on these objects. They are created when the database is created, and are updated during the course of normal operation. You cannot explicitly create or drop them, but you can query and view their contents using the catalog views.

Overview of Storage Objects

The following database objects let you define how data will be stored on your system, and how performance (related to accessing the data) can be improved:

- Table space
- Container
- Buffer pool

Table Spaces

A database is organized into parts called *table spaces*. A table space is a place to store tables. When creating a table, you can decide to have certain objects such as indexes and large object (LOB) data kept separately from the rest of the table data. A table space can also be spread over one or more physical storage devices. The following diagram shows some of the flexibility you have in spreading data over table spaces:

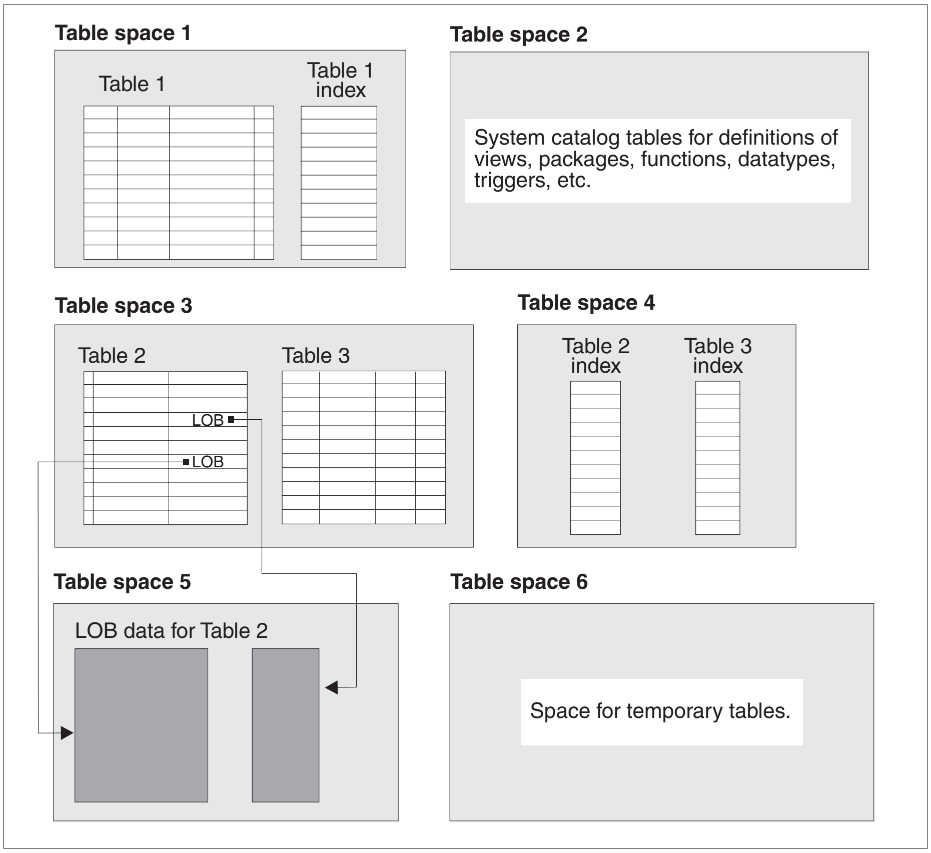


Figure 4. Table Spaces

Table spaces reside in nodegroups (see “Nodegroups” on page 9). Table space definitions and attributes are recorded in the database system catalog (see “System Catalog Tables” on page 11).

Containers are assigned to table spaces. A *container* is an allocation of physical storage (such as a file or a device).

A table space can be either system managed space (SMS), or database managed space (DMS). For an SMS table space, each container is a directory in the file space of the operating system, and the operating system’s file manager controls the storage space. For a DMS table space, each container is either a fixed size pre-allocated file, or a physical device such as a disk, and the database manager controls the storage space.

Figure 5 illustrates the relationship between tables, table spaces, and the two types of space. It also shows that tables, indexes, and long data are stored in table spaces.

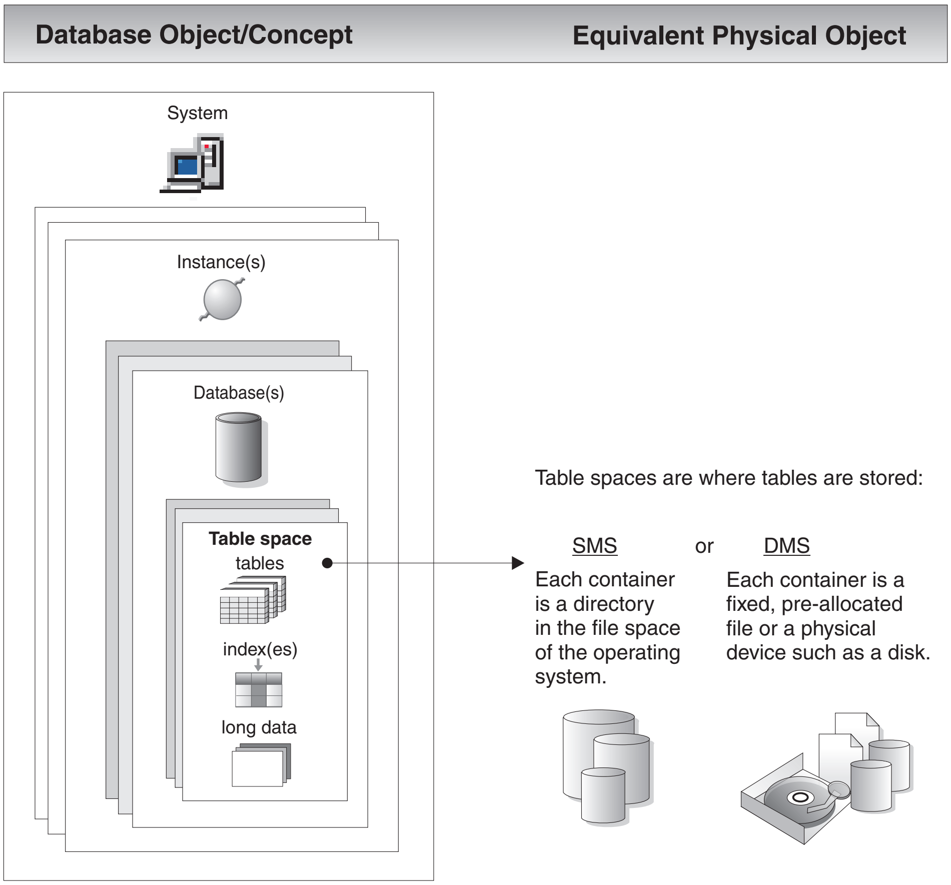

Database Object/Concept
Equivalent Physical Object
System
Instance(s)
Database(s)
Table space
tables
index(es)
long data
Table spaces are where tables are stored:
SMS
or
DMS
Each container is a directory in the file space of the operating system.
Each container is a fixed, pre-allocated file or a physical device such as a disk.

Figure 5. Table Spaces and Tables

Figure 6 on page 15 shows the three table space types: *regular*, *temporary*, and *long*.

Tables containing user data exist in regular table spaces. The default user table space is called USERSPACE1. Indexes are also stored in regular table spaces. The system catalog tables exist in a regular table space. The default system catalog table space is called SYSCATSPACE.

Tables containing long field data or long object data, such as multi-media objects, exist in long table spaces.

Temporary table spaces are classified as either system or user. *System temporary table spaces* are used to store internal temporary data required during SQL operations such as sorting, reorganizing tables, creating indexes, and joining tables. Although you can create any number of system temporary table spaces, it is recommended that you create only one, using the page size that the majority of your tables use. The default system temporary table space is called TEMPSPACE1. *User temporary table spaces* are used to store declared global temporary tables that store application temporary data. User temporary table spaces are *not* created by default at database creation time.

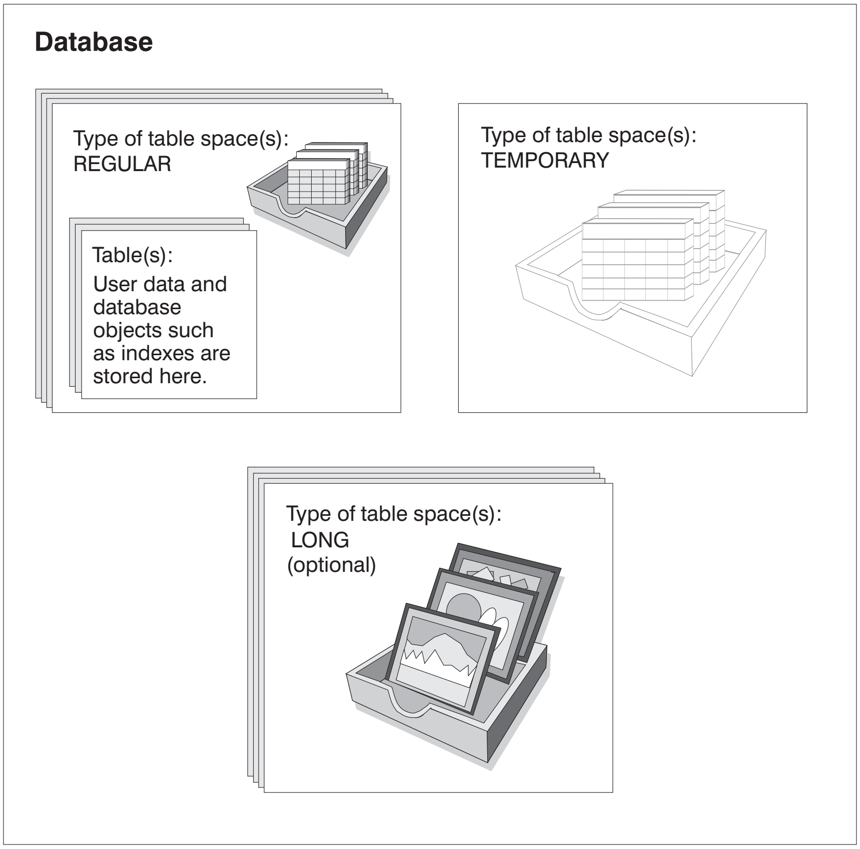

Database
Type of table space(s): REGULAR
Table(s): User data and database objects such as indexes are stored here.
Type of table space(s): TEMPORARY
Type of table space(s): LONG (optional)

Figure 6. Three Table Space Types

Containers

A *container* is a physical storage device. It can be identified by a directory name, a device name, or a file name.

A container is assigned to a table space. A single table space can span many containers, but each container can belong to only one table space.

Figure 7 illustrates the relationship between tables and a table space within a database, and the associated containers and disks.

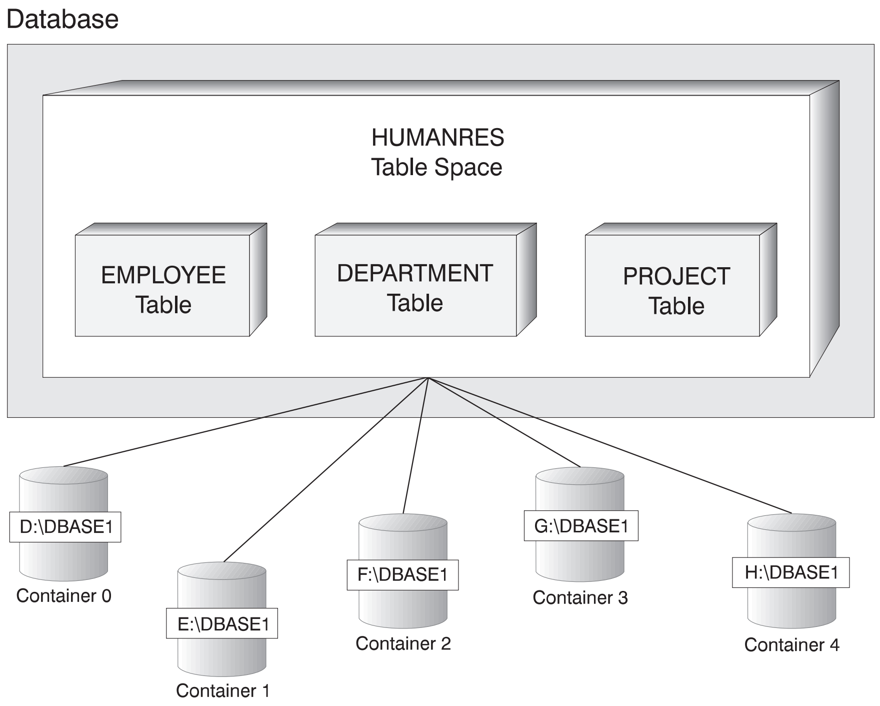


Figure 7. Table Spaces and Tables Within a Database

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

Data for any table will be stored on all containers in a table space in a round-robin fashion. This balances the data across the containers that belong to a given table space. The number of pages that the database manager writes to one container before using a different one is called the *extent size*.

Buffer Pool

A *buffer pool* is the amount of main memory allocated to cache table and index data pages as they are being read from disk, or being modified. The purpose of the buffer pool is to improve system performance. Data can be accessed much faster from memory than from disk; therefore, the fewer times the database manager needs to read from or write to a disk (I/O), the better the performance. (You can create more than one buffer pool, although for most situations only one is required.)

The configuration of the buffer pool is the single most important tuning area, because you can reduce the delay caused by slow I/O.

Figure 8 illustrates the relationship between a buffer pool and containers.

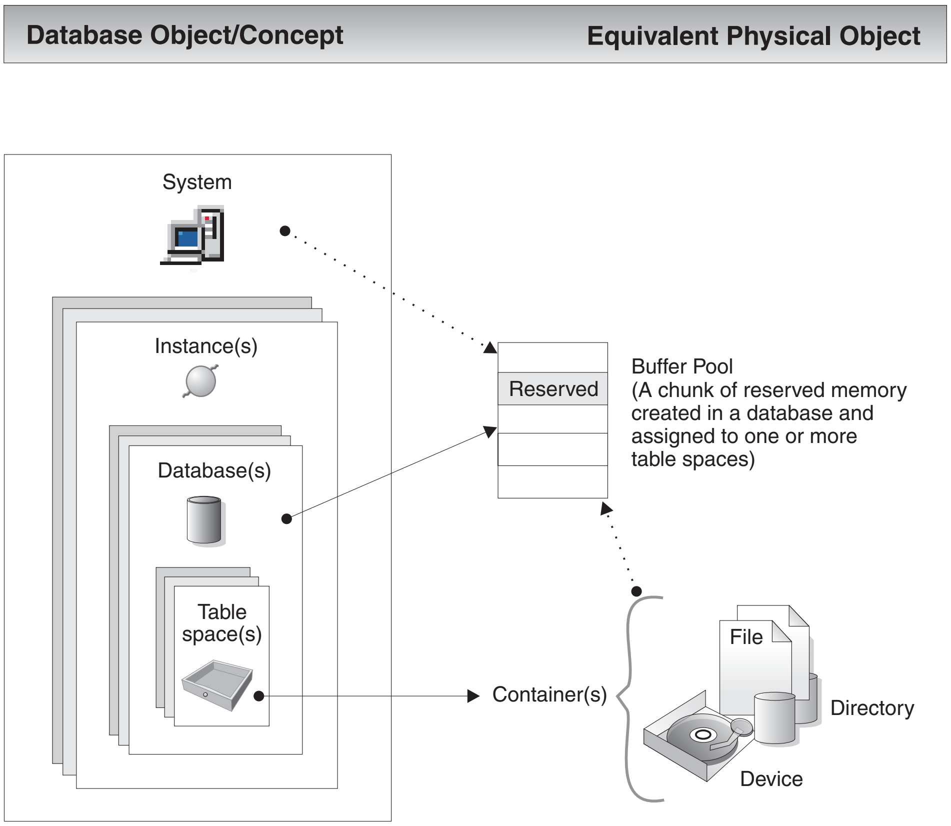

Database Object/Concept
Equivalent Physical Object
System
Instance(s)
Database(s)
Table space(s)
Reserved
Buffer Pool (A chunk of reserved memory created in a database and assigned to one or more table spaces)
Container(s)
File
Directory
Device

Figure 8. Buffer Pool and Containers

Overview of System Objects

When a DB2 instance or a database is created, a corresponding configuration file is created with default parameter values. You can modify these parameter values to improve performance.

Configuration Parameters

Configuration files contain parameters that define values such as the resources allocated to the DB2 products and to individual databases, and the diagnostic level. There are two types of configuration files: the database manager configuration file for each DB2 instance, and the database configuration file for each individual database (see Figure 9 on page 19).

The *database manager configuration file* is created when a DB2 instance is created. The parameters it contains affect system resources at the instance level, independent of any one database that is part of that instance. Values for many of these parameters can be changed from the system default values to improve performance or increase capacity, depending on your system's configuration.

There is one database manager configuration file for each client installation as well. This file contains information about the client enabler for a specific workstation. A subset of the parameters available for a server are applicable to the client.

A *database configuration file* is created when a database is created, and resides where that database resides. There is one configuration file per database. Its parameters specify, among other things, the amount of resource to be allocated to that database. Values for many of the parameters can be changed to improve performance or increase capacity. Different changes may be required, depending on the type of activity in a specific database.

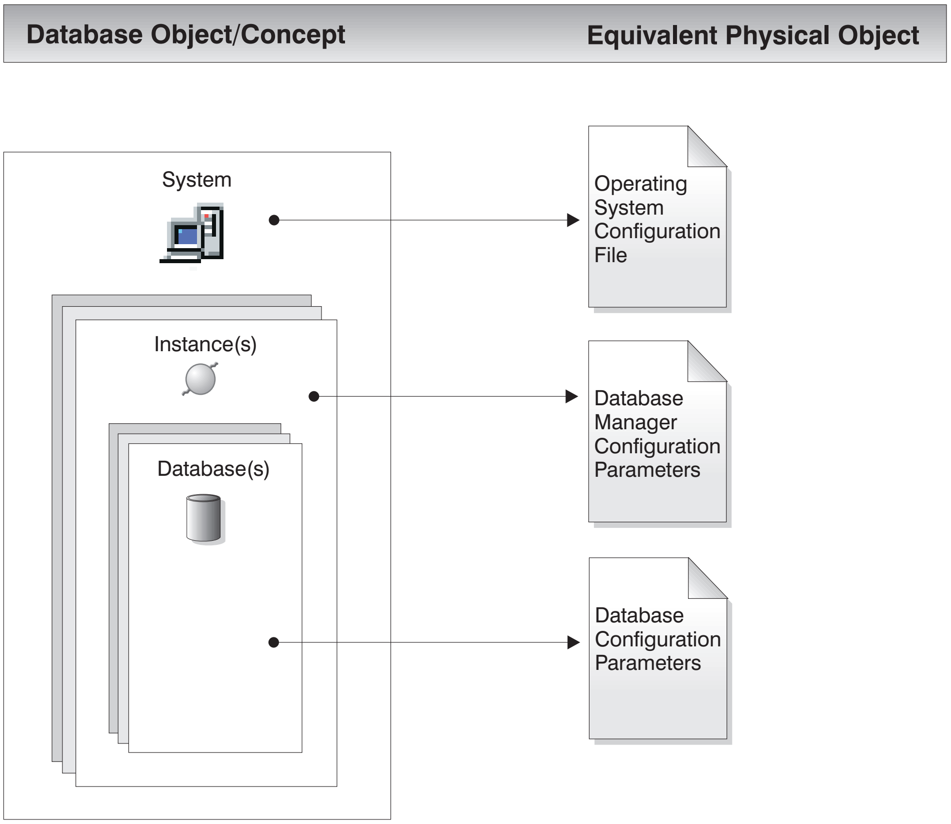

Database Object/Concept
Equivalent Physical Object
System
Instance(s)
Database(s)
Operating System Configuration File
Database Manager Configuration Parameters
Database Configuration Parameters

Figure 9. Configuration Parameter Files

Business Rules for Data

Within any business, data must often adhere to certain restrictions or rules. For example, an employee number must be unique. DB2 provides *constraints* as a way to enforce such rules.

DB2 provides the following types of constraints:

- NOT NULL constraint
- Unique constraint
- Primary key constraint
- Foreign key constraint
- Check constraint

NOT NULL constraint
NOT NULL constraints prevent null values from being entered into a column.

unique constraint
Unique constraints ensure that the values in a set of columns are unique and not null for all rows in the table. For example, a typical unique constraint in a DEPARTMENT table might be that the department number is unique and not null.

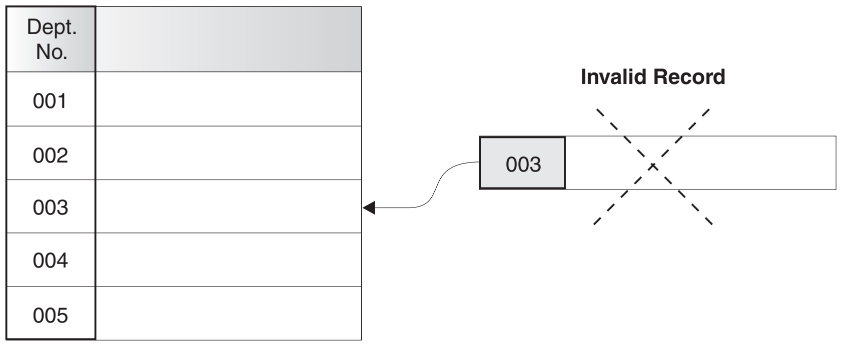


Figure 10. Unique Constraints Prevent Duplicate Data

The database manager enforces the constraint during insert and update operations, ensuring data integrity.

primary key constraint
Each table can have one primary key. A primary key is a column or combination of columns that has the same properties as a unique constraint. You can use a primary key and foreign key constraints to define relationships between tables.

Because the primary key is used to identify a row in a table, it should be unique and have very few additions or deletions. A table cannot have more than one primary key, but it can have multiple unique keys. Primary keys are optional, and can be defined when a table is created or altered. They are also beneficial, because they order the data when data is exported or reorganized.

In the following tables, DEPTNO and EMPNO are the primary keys for the DEPARTMENT and EMPLOYEE tables.

Table 1. DEPARTMENT Table

DEPTNO (Primary Key)	DEPTNAME	MGRNO
A00	Spiffy Computer Service Division	000010

Table 1. DEPARTMENT Table (continued)

DEPTNO (Primary Key)	DEPTNAME	MGRNO
B01	Planning	000020
C01	Information Center	000030
D11	Manufacturing Systems	000060

Table 2. 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

foreign key constraint

Foreign key constraints (also known as referential integrity constraints) enable you to define required relationships between and within tables.

For example, a typical foreign key constraint might state that every employee in the EMPLOYEE table must be a member of an existing department, as defined in the DEPARTMENT table.

To establish this relationship, you would define the department number in the EMPLOYEE table as the foreign key, and the department number in the DEPARTMENT table as the primary key.

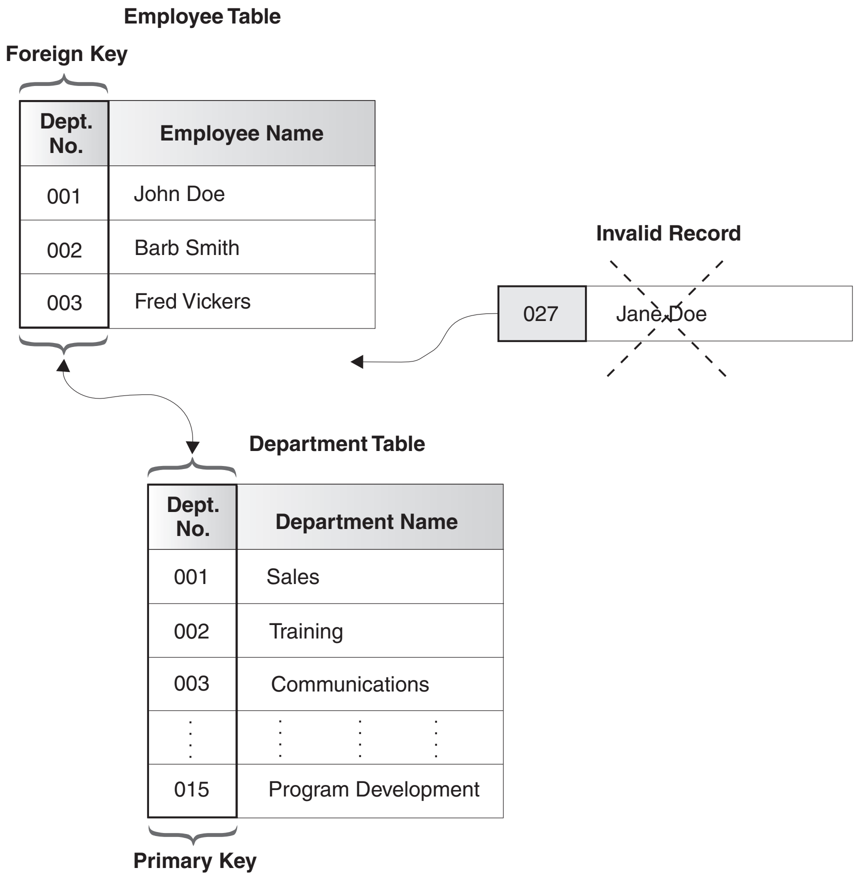


Figure 11. Foreign and Primary Key Constraints Define Relationships and Protect Data

check constraint

A check constraint is a database rule that specifies the values allowed in one or more columns of every row of a table.

For example, in an EMPLOYEE table, you can define the Type of Job column to be "Sales", "Manager", or "Clerk". With this constraint, any record with a different value in the Type of Job column is not valid, and would be rejected, enforcing rules about the type of data allowed in the table.

You can also use *triggers* in your database. Triggers are more complex and potentially more powerful than constraints. They define a set of actions that are executed in conjunction with, or triggered by, an INSERT, UPDATE, or DELETE clause on a specified base table. 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 be used in a banking application to raise an alert if a withdrawal from an account did not fit a customer's standard withdrawal patterns. For more information about triggers, refer to the *Application Development Guide*.

Recovering a Database

A database can become unusable because of hardware or software failure (or both), and different failure scenarios may require different recovery actions. You should have a rehearsed strategy in place to protect your database against the possibility of failure.

You can find in-depth information about backup, recovery and how to develop a good backup and recovery strategy in the *Data Recovery and High Availability Guide and Reference*.

The concept of a database backup is the same as any other data backup: taking a copy of the data and storing it on a different medium in case of failure or damage to the original. The simplest case of a backup involves shutting down the database to ensure that no further transactions occur, and then simply backing it up.

If your database is damaged or corrupted, you can rebuild it from the backup image. The rebuilding of the database is called recovery.

If a database crashes while transactions are in progress, a process called crash recovery will take place when the database is restarted. *Crash recovery* is the process by which the database is moved back 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.

If the database is damaged or corrupted, there are two ways to recover it: version recovery and rollforward recovery.

Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation. A database backup allows you to restore a database to a state identical to the one at the time that the backup was made. However, every unit of work from the time of the backup to the time of the failure is lost.

If you want to be able to restore a database past the point when the backup was taken, you must employ *rollforward recovery*. To use the rollforward recovery method, you must have taken a backup of the database, and archived the logs (by enabling either the *logretain* or the *userexit* database configuration parameters, or both).

Every database includes recovery logs, which are used to recover from application or system errors. In combination with the database backups, they can be used to recover the database right up to the point in time when the error occurred. Log files are created automatically when a database is created. You cannot directly modify log files.

Another important recovery object is the recovery history file. The recovery history file contains a summary of the backup information that can be used in case all or part of the database must be recovered to a given point in time. It is used to track recovery-related events such as backup, restore, and load operations.

Note: All of the information on backup and recovery, and the corresponding information from the Command Reference and the API Reference, has been consolidated in the *Data Recovery and High Availability Guide and Reference*. The *Data Recovery and High Availability Guide and Reference* is your primary, single source for backup and recovery information.

Reorganizing Tables in a Database

A table can become fragmented after many updates, causing performance to deteriorate. If you collected statistics and did not notice a visible performance improvement, reorganizing table data may help. When you reorganize table data, you are rearranging the data into a physical sequence according to a specified index, and removing the free space that is inherent in fragmented data. This can provide faster access to the data, thereby improving performance.

Before you reorganize tables, it is recommended that you invoke the REORGCHK command, and collect statistics on the table. Running this command will help you determine whether a reorganization of the table data is appropriate. Refer to the *Command Reference* for information about the REORGCHK command.

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 you must prove that you are who you say you are. 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, may not exist at all. On UNIX based systems, 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 the Windows 3.1 operating system.

The security facility requires two items to authenticate a user: a user ID and a password. The user ID identifies the user to the security facility. By supplying the correct password (information known only to the user and the security facility) the user's identity (corresponding to the user ID) is verified.

Once authenticated:

- The user must 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 UNIX based systems, a DB2 *authid* is derived by transforming to uppercase letters a UNIX user ID that follows DB2 naming conventions. 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 to which the user belongs 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 are added to the DB2 group list. No error is returned, 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 a successful security system login as evidence of identity, and allows:
  - 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. It uses successful validation of this pair by the security facility as evidence of identity and allows:
  - Use of remote connections where the server requires proof of authentication

– Use of operations where the user wants to run a command under an identity other than the identity used for login.

DB2 administrators can allow others to change passwords on AIX and Windows NT EEE systems through the profile registry variable DB2CHGPWD_EEE. The default value for this variable is `NOT SET` (disabled). DB2CHGPWD_EEE accepts 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 may result in passwords not being consistent across all nodes. That is, if you use the "change password" feature, your password will only be changed at the node to which you are connected.

DB2 UDB on AIX can 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.

For additional information about the system entry validation checking that is particularly relevant if you have remote clients accessing the database, see "Selecting an Authentication Method for Your Server" in the *Administration Guide: Implementation*.

Authorization

Authorization is the process whereby DB2 obtains information about an authenticated DB2 user, indicating the database operations that 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 to which the user belongs, are compared with the recorded permissions. Based on this comparison, DB2 decides whether to allow 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. *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; system authorities are associated with 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 that 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), but is not considered for static SQL. The exception to this general case occurs 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 the DB2 documentation, where applicable.

For more information, see ″Privileges, Authorities, and Authorization″ in the *Administration Guide: Implementation*.

Federated Database Authentication and Authorization Overview

Because a DB2 federated database system can access information in multiple database management systems, additional steps may be required to secure your data.

When planning your approach to authentication, consider the fact that users may 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 may be necessary if data sources require a user ID and password. For more information, see ″Federated Database Authentication Processing″ in the *Administration Guide: Implementation*.

Similarly, 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.

Chapter 3. Federated Systems

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 Universal Database 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 (in 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 under such a DBMS (such as recursive SQL) are run under 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 (beyond integrity and locking constraints) to Oracle or other DBMS objects.

A DB2 federated system consists of a DB2 UDB 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 30 for a visual representation of a federated database environment.

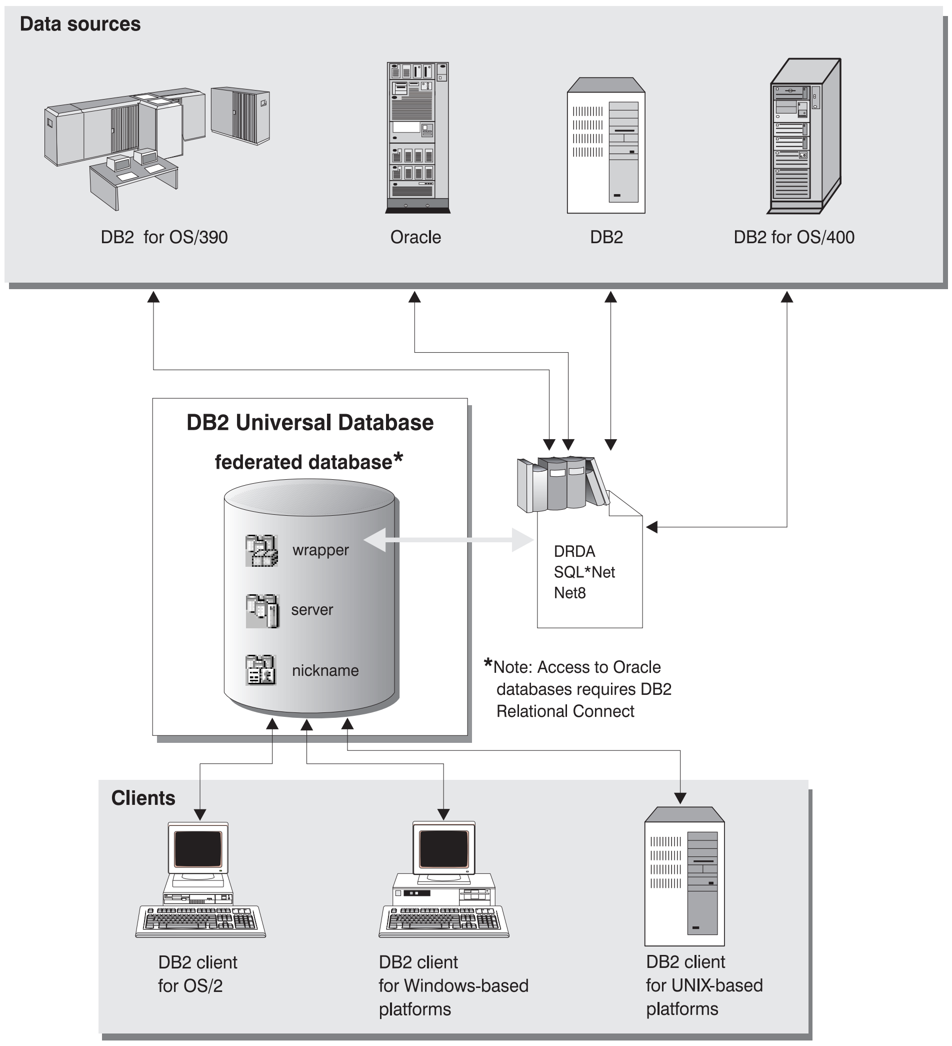


Figure 12. A Federated Database System

DB2 federated database catalog entries contain information about data source objects: what they are called, what 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 of the database. (One exception: a 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, and so on) 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 a DB2 type
- Function mappings, to map a local function to a data source function
- Index specifications, to improve performance.

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

There are many factors affecting federated system performance. The most critical factor 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. Refer to the *Administration Guide: Performance* for additional information about 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 limited 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 Enterprise Edition (EE) and DB2 Enterprise - Extended Edition (EEE) can support federated databases. To enable a federated system:

1. Select the *Distributed Join for DB2 Databases* installation option of DB2 EE or EEE during installation.
2. If including Oracle databases in your federated system, install DB2 Relational Connect. For more information, refer to the *Installation and Configuration Supplement*.
3. Set the database manager configuration parameter *federated* to "YES".
4. Create wrappers, servers, and nicknames (see "Creating a Database" in the *Administration Guide: Implementation* for more information).
5. Create additional objects, or set options as required (see "Implementing Your Design" in the *Administration Guide: Implementation* for more information).

Chapter 4. Parallel Database Systems

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 a 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 (see “Nodegroups” on page 9), 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 other rows 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 total data in the database.

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 issuing SQL statements.

User interaction occurs through one database partition, 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 13 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. (Nodegroup 3).
- 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.

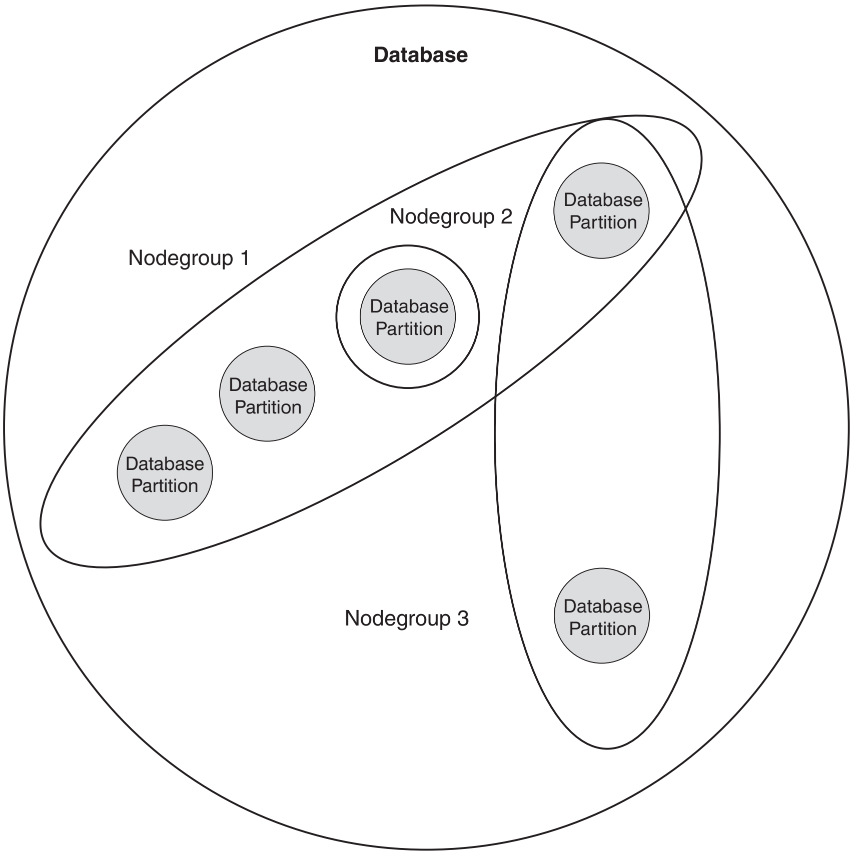


Figure 13. Nodegroups in a Database

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, see "Designing Nodegroups" on page 100.

Types of Parallelism

Components of a task, such as a database query, can be run in parallel to dramatically enhance performance. The nature of the task, the database configuration, and the hardware environment, all determine how DB2 will perform a task in parallel. These considerations are interrelated, and should be considered together when you work on the physical and logical design of a database. This section describes the following types of parallelism that are supported by DB2:

- I/O
- Query
- Utility

I/O Parallelism

When there are multiple containers for a table space, the database manager can exploit *parallel I/O*. Parallel I/O refers to the process of writing to, or reading from, two or more I/O devices simultaneously; it can result in significant improvements in throughput.

I/O parallelism is a component of each hardware environment described in "Hardware Environments" on page 38. Table 3 on page 46 lists the hardware environments best suited to 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 executes independently of the others, but DB2 executes all of them at the same time. DB2 has always supported this type of parallelism.

Intra-query parallelism refers to the simultaneous processing of parts of a single query, using either *intra-partition parallelism*, *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 38.)

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

Figure 14 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion. The pieces are copies of each other. To utilize intra-partition parallelism, you must configure the database appropriately. You can choose the degree of parallelism or let the system do it for you. The degree of parallelism represents the number of pieces of a query running in parallel.

Table 3 on page 46 lists the hardware environments best suited for intra-partition parallelism.

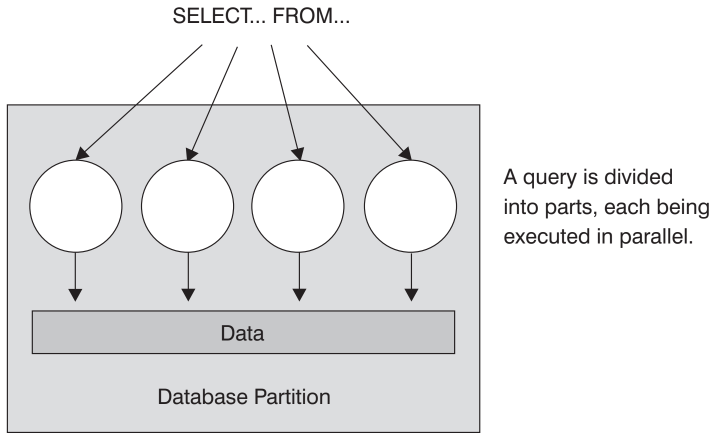

SELECT... FROM...
A query is divided into parts, each being executed in parallel.
Data
Database Partition

Figure 14. 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 run in parallel. (Some of the utilities also perform this type of parallelism. See “Utility Parallelism” on page 38.)

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

Figure 15 on page 37 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion on a single partition.

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

Table 3 on page 46 lists the hardware environments best suited for inter-partition parallelism.

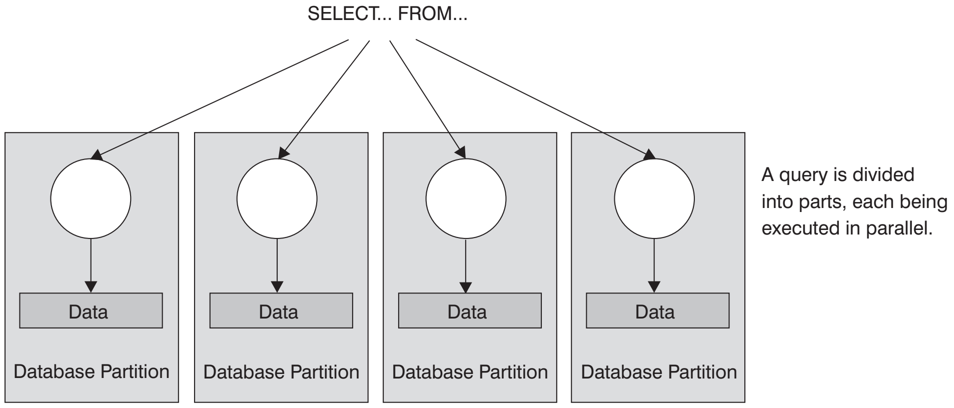


Figure 15. Inter-partition Parallelism

Simultaneous Intra-partition and Inter-partition Parallelism

You can use intra-partition parallelism and inter-partition parallelism at the same time. This combination provides two dimensions of parallelism, resulting in an even more dramatic increase in the speed at which queries are processed:

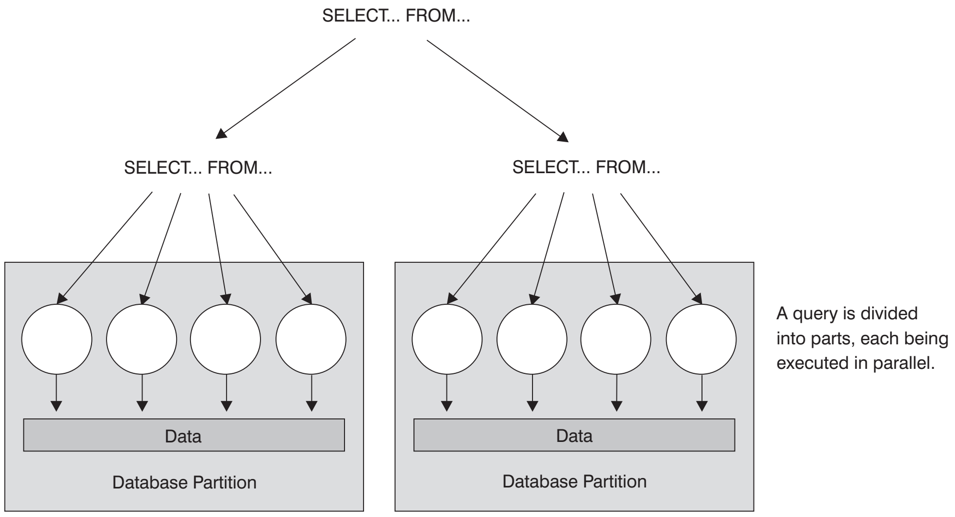


Figure 16. Simultaneous Inter-partition and Intra-partition Parallelism

Utility Parallelism

DB2 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 load utility can take advantage of intra-partition parallelism and I/O parallelism. Loading data is a CPU-intensive task. The load utility takes advantage of multiple processors for tasks such as parsing and formatting data. It can also use parallel I/O servers to write the data to containers in parallel. Refer to the *Data Movement Utilities Guide and 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 the LOAD command at each database partition where the table resides. Refer to the *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 statement 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 backup and restore operations. 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 utilities.

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, or MLN, in DB2 Parallel Edition for AIX Version 1)

Capacity and scalability are discussed for each environment. *Capacity* refers to the number of users and applications able to access the database. This is in large part determined by memory, agents, locks, I/O, and storage management. *Scalability* refers to the ability of 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 (see Figure 17). It is referred to by many different names, including stand-alone database, client/server database, serial database, uniprocessor system, and single node or non-parallel environment.

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

Table 3 on page 46 lists the types of parallelism best suited to take advantage of this hardware configuration.

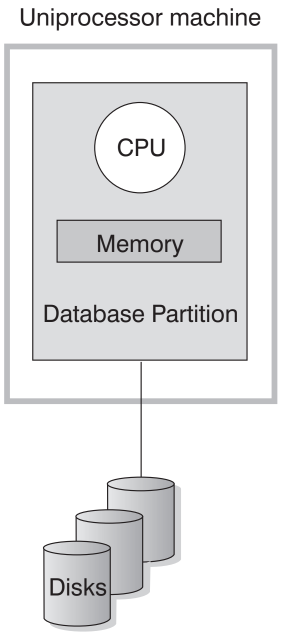


Figure 17. Single Partition On a Single Processor

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 take place at the same time.

A single-processor system is restricted by the amount of disk space the processor can handle. As workload increases, a single CPU may not be able to process user requests any faster, regardless of other 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.

Single Partition with Multiple Processors

This environment is typically made up of several equally powerful processors within the same machine (see Figure 18 on page 41), and is called a *symmetric multi-processor (SMP)* system. Resources, such as disk space and memory, are *shared*.

With multiple processors available, different database operations can be completed more quickly. DB2 can also divide the work of a single query among available processors to improve processing speed. Other database operations, such as loading data, backing up and restoring table spaces, and creating indexes on existing data, can take advantage of multiple processors.

Table 3 on page 46 lists the types of parallelism best suited to take advantage of this hardware configuration.

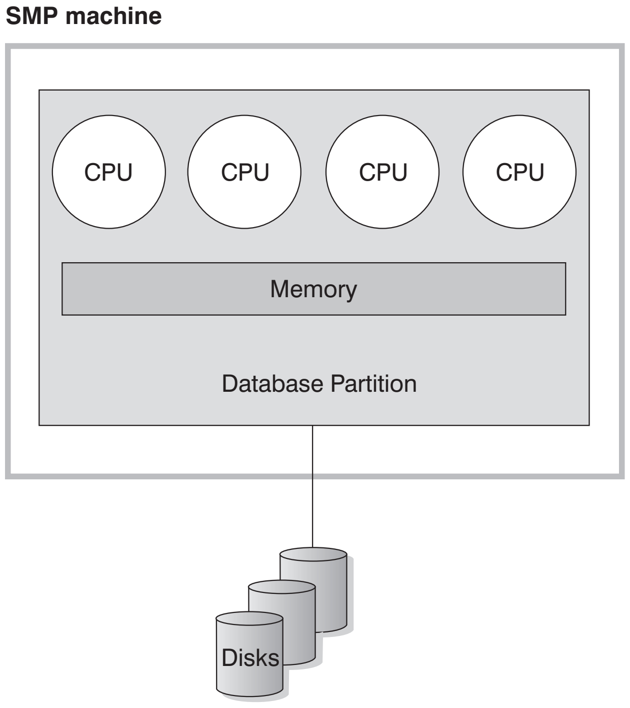


Figure 18. Single Partition Database Symmetric Multiprocessor System

Capacity and Scalability

In this environment you can add more processors. However, since the different processors may 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.

You can increase the I/O capacity of the database partition associated with your processor by increasing 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 scalability, you can consider moving to a system with multiple partitions.

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

Partitions with One Processor

In this environment, there are many database partitions. Each partition resides on its own machine, and has its own processor, memory, and disks (Figure 19 on page 43). All the machines are connected by a communications facility. This environment is referred to by many different names, including cluster, cluster of uniprocessors, massively parallel processing (MPP) environment, and 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.

A partitioned database environment allows a database to remain a logical whole, despite being physically divided across more than one partition. The fact that data is partitioned remains transparent to most users. Work can be divided among the database managers; each database manager in each partition works against its own part of the database.

Table 3 on page 46 lists the types of parallelism best suited to take advantage of this hardware configuration.

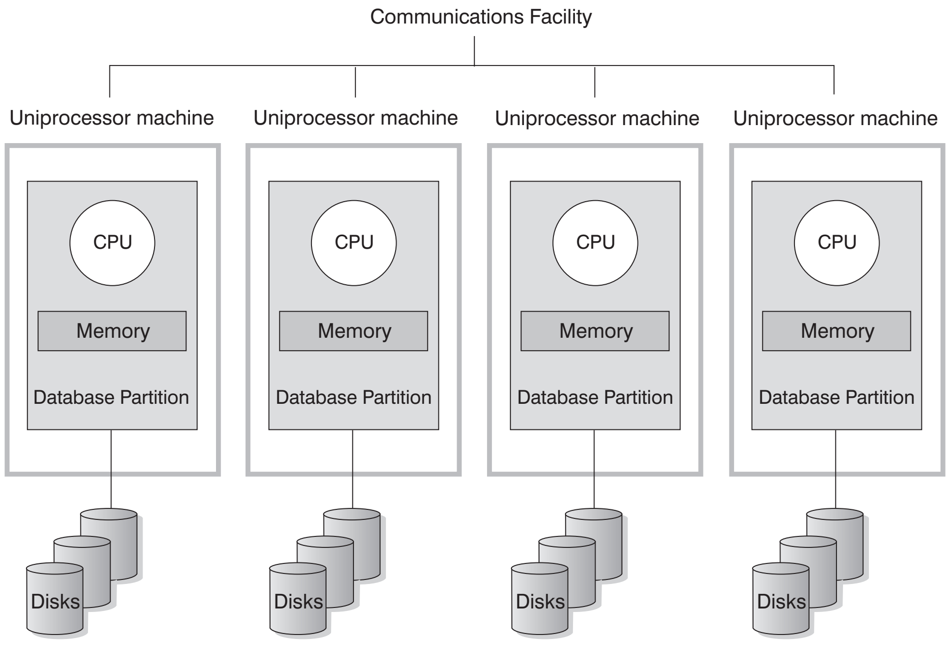


Figure 19. Massively Parallel Processing System

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 number is 512 nodes. However, there may be practical limits on 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.

Partitions with Multiple Processors

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* (Figure 20 on page 44).

This configuration combines the advantages of SMP and MPP parallelism. This means that 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.

Table 3 on page 46 lists the types of parallelism best suited to take advantage of this hardware configuration.

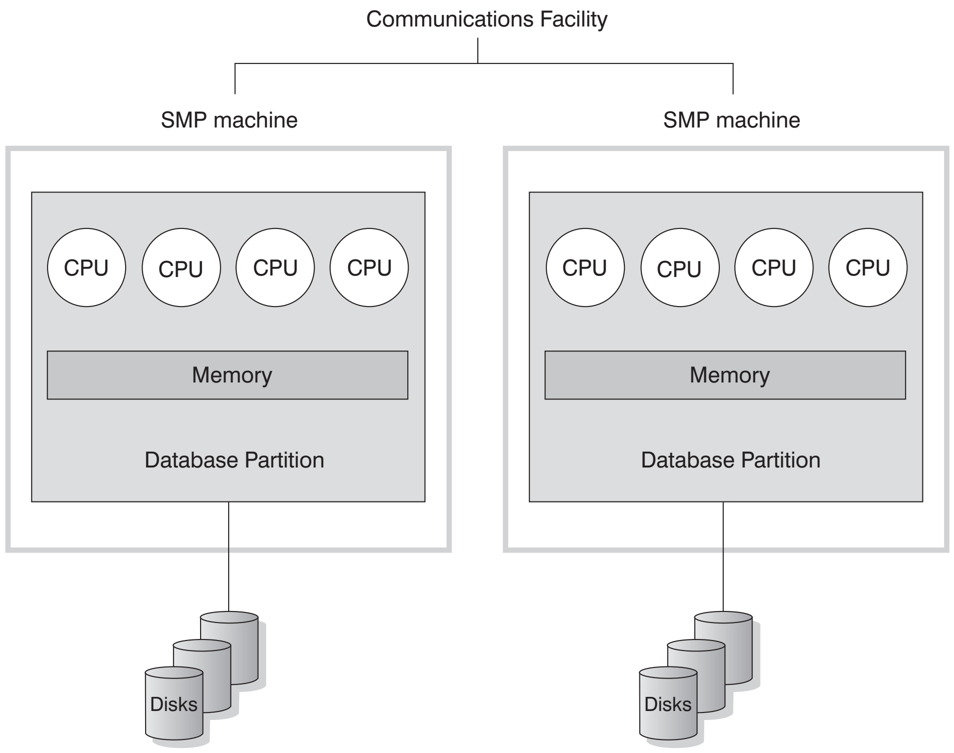


Figure 20. Cluster of SMPs

Capacity and Scalability: In this environment you can add more database partitions, and you can 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, database partitions do not share the resources. Processors are shared but disks and memory are not.

Logical database partitions provide scalability. Multiple database managers running on multiple logical partitions may make fuller use of available resources than a single database manager could. Figure 21 on page 45 illustrates the fact that you may gain more scalability on an SMP machine by adding more partitions; this is particularly true for machines with many processors. By partitioning the database, you can administer and recover each partition separately.

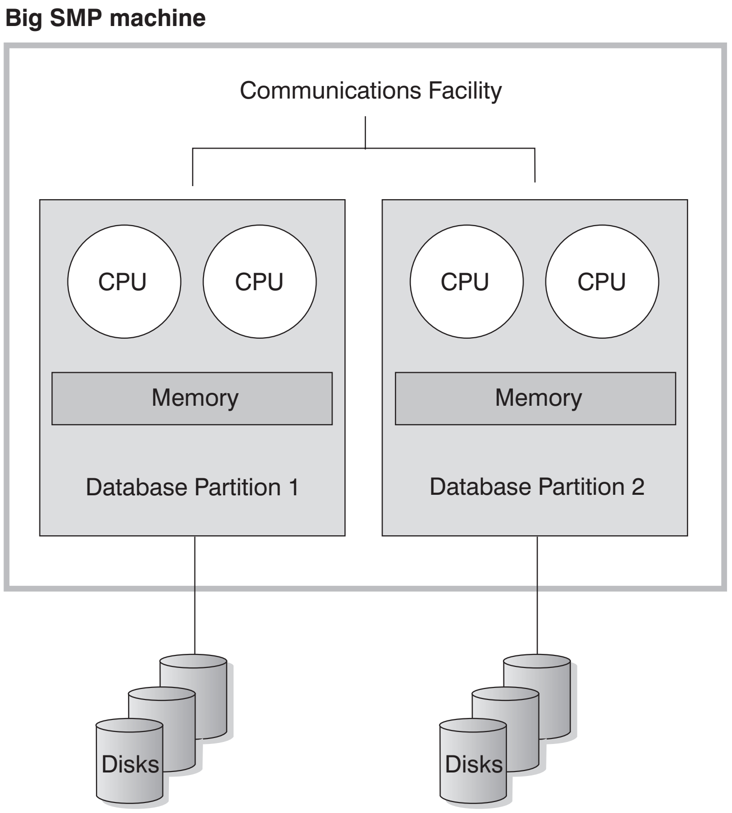


Figure 21. Partitioned Database, Symmetric Multiprocessor System

Figure 22 on page 46 illustrates the fact that you can multiply the configuration shown in Figure 21 to increase processing power.

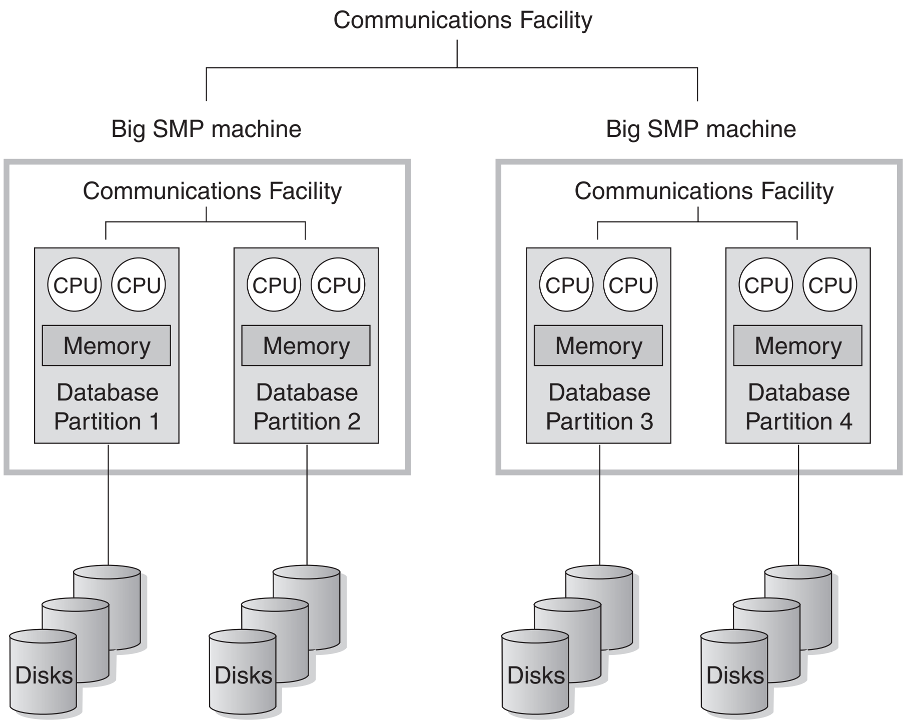


Figure 22. Partitioned Database, Symmetric Multiprocessor Systems Clustered Together

Table 3 lists the types of parallelism best suited to take advantage of this hardware environment.

Note: 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. Upon machine failure, a database partition can be automatically moved and restarted on a second machine that already contains another partition of the same database. For more information, see "Chapter 11. Introducing High Availability and Failover Support" on page 179.

Summary of Parallelism Best Suited to Each Hardware Environment

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

Table 3. Types of Parallelism Possible in Each Hardware Environment

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

Table 3. Types of Parallelism Possible in Each Hardware Environment (continued)

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

Chapter 5. About Data Warehousing

DB2 Universal Database offers the Data Warehouse Center, a component that automates data warehouse processing. You can use the Data Warehouse Center to define the data to include in the warehouse. Then, you can use the Data Warehouse Center to automatically schedule refreshes of the data in the warehouse.

This section provides an overview of data warehousing and data warehousing tasks. For more detailed information about warehousing, and for information on using the Data Warehouse Center, refer to the *Data Warehouse Center Administration Guide* and the Data Warehouse Center online help.

What is Data Warehousing?

The systems that contain *operational data*—the data that runs the daily transactions of your business—contain information that is useful to business analysts. For example, analysts can use information about which products were sold in which regions at which time of year to look for anomalies or to project future sales.

However, there are several problems with analysts accessing the operational data directly:

- They might not have the expertise to query the operational database. For example, querying IMS databases requires an application program that uses a specialized type of data manipulation language. In general, those programmers who have the expertise to query the operational database have a full-time job in maintaining the database and its applications.
- Performance is critical for many operational databases, such as databases for a bank. The system cannot handle users making ad hoc queries.
- The operational data generally is not in the best format for use by business analysts. For example, sales data that is summarized by product, region, and season is much more useful to analysts than the raw data.

Data warehousing solves these problems. In *data warehousing*, you create stores of *informational data*—data that is extracted from the operational data, and then transformed for end-user decision making. For example, a data warehousing tool might copy all the sales data from the operational database, perform calculations to summarize the data, and write the summarized data to a target in a separate database from the operational data. End users can query the separate database (the *warehouse*) without impacting the operational databases.

The following sections describe the objects (subject areas, warehouse sources, warehouse targets, agents, agent sites, steps, and processes) that you will use to create and maintain your data warehouse.

Subject Areas

A *subject area* identifies and groups the processes that relate to a logical area of the business. For example, if you are building a warehouse of marketing and sales data, you can define a Sales subject area and a Marketing subject area. You can then add the processes that relate to sales underneath the Sales subject area. Similarly, you can add the definitions that relate to the marketing data underneath the Marketing subject area.

Warehouse Sources

Warehouse sources identify the tables and files that will provide data to your warehouse. The Data Warehouse Center uses the specifications in the warehouse sources to access and select the data. The sources can be nearly any relational or nonrelational source (table, view, or file) that has connectivity to your warehouse.

Warehouse Targets

Warehouse targets are database tables or files that contain data that has been transformed so that end users can use it. Like a warehouse source, warehouse targets can also provide data to Data Warehouse Center steps.

Warehouse Agents and Agent Sites

Data Warehouse Center *agents* manage the flow of data between the data sources and the target warehouses. Agents are available on the Windows NT, AIX, OS/2, OS/390, OS/400, and SUN Solaris operating systems. The agents use Open Database Connectivity (ODBC) drivers or DB2 CLI to communicate with different databases.

Several agents can handle the transfer of data between sources and target warehouses. The number of agents that you use depends on your existing connectivity configuration and the volume of data that you plan to move through your warehouse. Additional instances of an agent can be generated if multiple processes that require the same agent are running simultaneously.

Agents can be local or remote. A *local warehouse agent* is an agent that is installed on the same machine as the warehouse server. A *remote warehouse agent* is an agent that is installed on another machine that has connectivity to the warehouse server.

An *agent site* is a logical name for a workstation where agent software is installed. The agent site name is not the same as the TCP/IP host name. A single physical machine can have only one TCP/IP host name. However, you can define multiple agent sites on a single machine. A logical name identifies each agent site.

The *default agent site*, named the Default VW AgentSite, is a local agent on Windows NT that Data Warehouse Center defines during initialization of the warehouse control database.

Steps and Processes

A *step* is a logical entity in the Data Warehouse Center that defines:

- The structure of the output table or file.
- The mechanism (either SQL or a program) for populating the output table or file.
- The schedule by which the output table or file is populated.

Steps move data and transform data by using SQL statements or by calling programs. When you run a step, the transfer of data between the warehouse source and the warehouse target, and any transformation of that data, takes place.

A *process* contains a series of steps that perform transformation and movement tasks. In general, a process populates a warehouse target in a warehouse database by extracting data from one or more warehouse sources, which can be database tables or files. However, you can also define a process for launching programs that does not specify any warehouse sources or targets.

You can run a step on demand, or you can schedule a step to run at a set time. You can schedule a step to run one time only, or you can schedule it to run repeatedly, such as every Friday. You can also schedule steps to run in sequence, so that when one step finishes running, the next step begins running. You can schedule steps to run upon (successful or unsuccessful) completion of another step. If you schedule a process, the first step in the process runs at the scheduled time.

When a step or a process runs, it can save data by:

- Replacing all the data in the warehouse target with new data.
- Appending the new data to the existing data.
- Appending a separate edition of data.

Suppose that you want Data Warehouse Center to perform the following tasks:

1. Extract data from different databases.
2. Convert the data to a single format.
3. Write the data to a table in a data warehouse.

You would create a process that contained individual steps. Each step would perform a separate task, such as extracting the data from the databases, or converting it to the correct format. You would then use another step to populate the target table, which contains the transformed data.

The following sections describe the various types of steps that you will find in the Data Warehouse Center. For more information about steps, refer to the *Data Warehouse Center Administration Guide*.

SQL Steps

An SQL step uses an SQL SELECT statement to extract data from a warehouse source, and generates an INSERT statement to insert the data into the warehouse target table.

Program Steps

There are several types of program steps: DB2 for AS/400 Programs, DB2 for OS/390 Programs, DB2 for UDB Programs, Visual Warehouse 5.2 DB2 Programs, OLAP Server Programs, File Programs, and Replication. These steps run predefined programs and utilities.

Transformer Steps

Transformer steps are stored procedures and user-defined functions that specify statistical or warehouse transformers that you can use to transform data. You can use transformers to clean, invert, and pivot data; generate primary keys and period tables; and calculate various statistics.

In a transformer step, you specify one of the statistical or warehouse transformers. When you run the process, the transformer step writes data to one or more warehouse targets.

User-defined Program Steps

A *user-defined program step* is a logical entity within the Data Warehouse Center that represents an application that you want the Data Warehouse Center to start. A warehouse agent can start a user-defined program step:

- During the population of a warehouse target.
- After the population of a warehouse target.
- By itself.

For example, you can write a user-defined program that will perform the following process:

1. Export data from a table.
2. Manipulate that data.
3. Write the data to an interim output resource or a warehouse target.

Warehousing Tasks

Creating a data warehouse involves the following tasks:

- Defining a subject area that identifies and groups the processes that you will use in your warehouse.
- Exploring the source data (or operational data), and defining warehouse sources.
- Creating a database to use as the warehouse, and defining warehouse targets.
- Specifying how to move and transform the source data into its format for the warehouse database by defining a process.
- Testing the steps that you have defined, and scheduling them to run automatically.
- Administering the warehouse by defining security, and monitoring database usage.
- Creating an information catalog of the data in the warehouse, if you have the DB2 Warehouse Manager package. An information catalog is a database that contains business metadata. The metadata helps users identify and locate data and information available to them in the organization. End users of the warehouse can search the catalog to determine which tables to query.
- Defining a star schema model for the data in the warehouse. A star schema is a specialized design that consists of multiple dimension tables (which describe aspects of a business) and one fact table (which contains the facts about the business). For example, if you manufacture soft-drinks, some dimension tables are products, markets, and time. The fact table contains transaction information about the products that are ordered in each region by season.
- Joining the fact table and dimension tables to combine details from the dimension tables with the order information. For example, you could join the product dimension with the fact table to add information about how each product was packaged to the orders.

You can learn more about these tasks and others by using the *Business Intelligence Tutorial*, viewing the *DB2 Universal Database Quick Tour*, or reading the *Data Warehouse Center Administration Guide*.

Chapter 6. About Spatial Extender

This section introduces Spatial Extender by explaining its purpose and discussing the data that it processes. For detailed information about using Spatial Extender, refer to the *Spatial Extender User's Guide and Reference*.

The Purpose of Spatial Extender

Use Spatial Extender to create a *geographic information system* (GIS): a complex of objects, data, and applications that allows you to generate and analyze spatial information about geographic features. *Geographic features* include the objects that form the surface of the earth and the objects that occupy it. They make up both the natural environment (examples are rivers, forests, hills, and deserts) and the cultural environment (cities, residences, office buildings, landmarks, and so on).

Spatial information includes facts such as:

- The location of geographic features with respect to their surroundings (for example, points within a city where hospitals and clinics are located, or the proximity of the city's residences to local earthquake zones)
- Ways in which geographic features are related to each other (for example, information that a certain river system is enclosed within a specific region, or that certain bridges in that region cross over the river system's tributaries)
- Measurements that apply to one or more geographic features (for example, the distance between an office building and its lot line, or the length of a bird preserve's perimeter)

Spatial information, either by itself or in combination with traditional database output, can help you to design projects and make business and policy decisions. For example, suppose that a welfare district manager needs to verify which welfare applicants and recipients actually live within the area that the district services. Spatial Extender can derive this information from the serviced area's location and from the addresses of the applicants and recipients.

Or suppose that the owner of a restaurant chain wants to do business in nearby cities. To determine where to open new restaurants, the owner needs answers to such questions as: Where in these cities are there concentrations of the type of people who would frequent my restaurants? Where are the major highways? Where is the crime rate lowest? Where are my competitor's restaurants located? Spatial Extender can produce spatial information in visual displays to answer such questions, and the underlying database management system can generate labels and text to explain the displays.

Data that Represents Geographic Features

This section provides an overview of the data that you generate, store, and operate upon to obtain spatial information. The topics covered are:

- How data represents geographic features
- The nature of spatial data
- Ways to produce spatial data

How Data Represents Geographic Features

In Spatial Extender, a geographic feature can be represented by a row in a table or view, or by a portion of such a row. For example, consider the following two geographic features: office buildings and residences. In Figure 23, each row of the BRANCHES table represents a branch office of a bank. As a variation, each row of the CUSTOMERS table, taken as a whole, represents a customer of the bank; but part of each row—specifically, the cells that contain a customer's address—can be viewed as representing the customer's residence.

These tables contain data that identifies and describes the bank's branches and customers. Such data is called *attribute data*.

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141	A	A

Figure 23. Table row that represents a geographic feature; table row whose address data represents a geographic feature. The row of data in the BRANCHES table represents a branch office of a bank. The cells for address data in the CUSTOMERS table represent the residence of a customer.

A subset of the attribute data (the values that represent branch and customer addresses) can be translated into values that yield spatial information. For example, as shown in the figure, one branch office address is 92467 Airzone Blvd., San Jose CA 95141. A customer address is 9 Concourt Circle, San Jose CA 95141. Spatial Extender can translate these addresses into values that indicate where the branch and the customer's home are situated with respect to their respective surroundings. Figure 24 shows the BRANCHES and CUSTOMERS tables with new columns that contain such values.

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141		A	A

Figure 24. Tables with spatial columns added. In each table, the LOCATION column will contain coordinates that correspond to the addresses.

When addresses and similar identifiers are used as the starting point for spatial information, they are called *source data*. Because the values derived from them yield spatial information, these derived values are called *spatial data*. The next section describes spatial data and introduces its associated data types.

The Nature of Spatial Data

Much spatial data is made up of coordinates. A *coordinate* is a number that denotes a position that is relative to a point of reference. For example, latitudes are coordinates that denote positions relative to the equator. Longitudes are coordinates that denote positions relative to the Greenwich meridian. Thus, the position of Yellowstone National Park is defined by its latitude (44.45 degrees north of the equator) and its longitude (110.40 degrees west of the Greenwich meridian).

Latitudes, longitudes, their points of reference, and other associated parameters are referred to collectively as a *coordinate system*. Coordinate systems based on values other than latitude and longitude also exist. These coordinate systems have their own measures of position, points of reference, and additional distinguishing parameters.

The simplest spatial data item consists of two coordinates that define the position of a single geographic feature. The term *data item* refers to the value or values that occupy a cell in a relational table. A more extensive spatial data item consists of several coordinates that define a linear path, such as a road or a river. A third kind consists of coordinates that define the perimeter of an area, such as the rim of a land parcel or flood plain.

Each spatial data item is an instance of a spatial data type. The data type for two coordinates that mark a location is ST_Point; the data type for coordinates that define linear paths is ST_LineString; and the data type for coordinates that define perimeters is ST_Polygon. These types, together with the other data types for spatial data, are structured types that belong to a single hierarchy.

Where Spatial Data Comes From

You can obtain spatial data by:

- Deriving it from attribute data
- Deriving it from other spatial data
- Importing it

Using Attribute Data as Source Data

Deriving spatial data from attribute data (such as addresses) is called *geocoding*. Figure 24 on page 57 shows two columns, one in the BRANCHES table and one in the CUSTOMERS table, designated for spatial data. Imagine that Spatial Extender geocodes the addresses in these tables and places the resulting output (coordinates that correspond to the addresses) into the columns. Figure 25 illustrates this result.

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	1653 3094

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141	953 1527	A	A

Figure 25. Tables that include spatial data derived from source data. The LOCATION column in the CUSTOMERS table contains coordinates that a geocoder derived from the address in the ADDRESS, CITY, STATE, and ZIP columns. Similarly, the LOCATION column in the BRANCHES table contains coordinates that the geocoder derived from the address in this table's ADDRESS, CITY, STATE, and ZIP columns.

Spatial Extender uses a function, called a *geocoder*, to geocode attribute data and place the resulting spatial data into columns.

Using Other Spatial Data as Source Data

Spatial data can be generated not only from attribute data, but also from other spatial data. For example, suppose that the bank whose branches are defined in the BRANCHES table wants to know how many customers are located within five miles of each branch. Before the bank can obtain this information from the database, it must supply the database with the definition of a zone that lies within a five-mile radius around each branch. A Spatial Extender function, ST_Buffer, can create such a definition. Using the coordinates of each branch as input, this function can generate the coordinates that demarcate the perimeters of the desired zones. Figure 26 shows the BRANCHES table with information supplied by ST_Buffer.

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	SALES_AREA
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	1653 3094	1002 2001, 1192 3564, 2502 3415, 1915 3394, 1002 2001

Figure 26. Table that includes new spatial data derived from existing spatial data. The coordinates in the SALES_AREA column were derived by the ST_Buffer function from the coordinates in the LOCATION column.

In addition to ST_Buffer, Spatial Extender provides several other functions that derive new spatial data from existing spatial data.

Importing Spatial Data

A third way to obtain spatial data is to import it from files that are in one of the formats that Spatial Extender supports. These files contain data that is usually applied to maps: census tracks, flood plains, earthquake faults, and so on. By using such data in combination with spatial data that you produce, you can augment the geographic information available to you. For example, if a public works department needs to determine the hazards to which a residential community is vulnerable, it could use ST_Buffer to define a zone around the community, and then import data on flood plains and earthquake faults to see which of these problem areas overlap the zone.

Part 3. Database Design

Chapter 7. Logical Database Design

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 65
- “Provide Column Definitions for All Tables” on page 67
- “Identify One or More Columns as the Primary Key” on page 70
- “Ensure that Equal Values Represent the Same Entity” on page 73
- “Consider Normalizing Your Tables” on page 74
- “Planning for Constraints Enforcement” on page 79
- “Other Database Design Considerations” on page 86.

Your goal in designing a database is to produce a representation of your environment that is easy to understand and that will serve as a basis for expansion. In addition, you want a database design that will help you maintain consistency and integrity of 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 8. Physical Database Design” on page 89, and in ″Implementing Your Design″ in the *Administration Guide: Implementation*.

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 represented as *tables*.

An *entity* is a person, object, or concept about which you want to store information. Some of the entities described in the sample tables are employees, departments, and projects. (For a description of the sample database, refer to the *SQL Reference*.)

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 4. 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. You may want to consider attributes to support multimedia objects such as documents, video or mixed media, image, and voice.

Within 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.

″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 the entity relationships within your enterprise, you also need to identify other types of information, such as the business rules that apply to that data.

Define Tables for Each Type of Relationship

Several types of relationships can be defined in a database. 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; this relationship is *multi-valued* for departments. The relationship between employees (single-valued) and departments (multi-valued) is a *one-to-many* relationship. The following types of relationships are discussed in this section:

- "One-to-Many and Many-to-One Relationships"
- "Many-to-Many Relationships" on page 66
- "One-to-One Relationships" on page 67

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

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

1. Group all the relationships for which the ″many″ side of the relationship is the same entity.
2. Define a single table for all the relationships in the group.

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

Table 5. 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 on the ″many″ side, so we define a department table, DEPARTMENT.

The following shows how these relationships are represented in tables:

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

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 shows how a many-to-many relationship (an employee can work on many projects, and a project can have many employees working on it) is represented in a table:

The employee activity (EMP_ACT) table:

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

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 shows how a one-to-one relationship is represented in a table:

The DEPARTMENT table:

DEPTNO	MGRNO
A00	000010
B01	000020
D11	000060

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 for that table. Selecting column names is described in detail in “Appendix A. Naming Rules” on page 187.

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

   The *data type* and *length* specify the type of data and the maximum length that are 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*.

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 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 an employee's resume.
- A *double-byte character large object* (DBCLOB) string, where the sequence of characters is 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*.

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 and share characteristics with existing types, but that are nevertheless considered to be separate and incompatible.

A *structured type* is a user-defined type whose structure 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) can 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 SQL functions, 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 represent shoe size, but they are incompatible, because the measurement base is different and cannot be compared. A user-defined function can be invoked to convert one shoe size to another.

For a better understanding of user-defined types, structured types, reference types, and user-defined functions, refer to the *SQL Reference*.

3. State which columns might need default values.

   Some columns cannot have meaningful values in all rows because:
   - A column value 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 is not yet known.

     For 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 not applicable), 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*.

Identify One or More Columns as the Primary Key

A *key* is a set of columns that can be used to identify or access a particular row or rows. The key is identified in the description of a table, index, or referential constraint. The same column can be part of more than one key.

A *unique key* is a key that is constrained so that no two of its values are equal. The columns of a unique key cannot contain NULL values. 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 mechanism used to enforce the uniqueness of the key is called a *unique index*. The unique index of a table is a column, or an ordered collection of columns, for which each value identifies (functionally determines) a unique row. A unique index can contain NULL values.

The *primary key* is one of the unique keys defined on a table, but is selected to be the key of first importance. There can be only 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. (You can also define indexes on non-primary key columns to efficiently access data when processing queries.)

If a table does not have a ″natural″ unique key, or if arrival sequence is the method used to distinguish unique rows, using a time stamp as part of the key can be helpful. (See also “Defining Identity Columns” on page 72.)

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, including its primary key column.

Table 6. 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. 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 defined, you may consider creating a new column that has unique values.

The following example shows a primary key containing more than one column (a composite key):

Table 7. 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 2 on page 21, there appear to be many candidate keys. The EMPNO, the PHONENO, and the LASTNAME columns each uniquely identify the employee.

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

- Persistence means that a primary key value for each row always exists.
- Uniqueness means that the key value for each row is different from all the others.
- Stability means that primary key values never change.

Of the three candidate keys in the example, only EMPNO satisfies all of these criteria. An employee may not have a phone number when joining a company. Last names can change, and, although they may be unique at one point, are not guaranteed to be so. The employee number column is the best 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, values in the employee number column are persistent.

Defining Identity Columns

An *identity column* provides a way for DB2 to automatically generate a unique numeric value for each row in a table. A table can have a single column that is defined with the identity attribute. Examples of an identity column include order number, employee number, stock number, and incident number.

Values for an identity column can be generated always or by default.

- An identity column that is defined as *generated always* is guaranteed to be unique by DB2. Its values are always generated by DB2; applications are not allowed to provide an explicit value.
- An identity column that is defined as *generated by default* gives applications a way to explicitly provide a value for the identity column. If a value is not given, DB2 generates one, but cannot guarantee the uniqueness of the value in this case. DB2 guarantees uniqueness only for the set of values that it generates. Generated by default is meant to be used for data propagation, in which the contents of an existing table are copied, or for the unloading and reloading of a table.

Identity columns are ideally suited to the task of generating unique primary key values. Applications can use identity columns to avoid the concurrency and performance problems that can result when an application generates its own unique counter outside of the database. For example, one common application-level implementation is to maintain a 1-row table containing a counter. Each transaction locks this table, increments the number, and then commits; that is, only one transaction at a time can increment the counter. In contrast, if the counter is maintained through an identity column, much higher levels of concurrency can be achieved because the counter is not locked by transactions. One uncommitted transaction that has incremented the counter will not prevent subsequent transactions from also incrementing the counter.

The counter for the identity column is incremented (or decremented) independently of the transaction. If a given transaction increments an identity counter two times, that transaction may see a gap in the two numbers that are generated because there may be other transactions concurrently incrementing the same identity counter (that is, inserting rows into the same table). If an application must have a consecutive range of numbers, that application should take an exclusive lock on the table that has the identity column. This decision must be weighed against the resulting loss of concurrency. Furthermore, it is possible that a given identity column can appear to have generated gaps in the number, because a transaction that generated a value for the identity column has rolled back, or the database that has cached a range of values has been deactivated before all of the cached values were assigned.

The sequential numbers that are generated by the identity column have the following additional properties:

- The values can be of any exact numeric data type with a scale of zero; that is, SMALLINT, INTEGER, BIGINT, or DECIMAL with a scale of zero. (Single- and double-precision floating-point are considered to be approximate numeric data types.)
- Consecutive values can differ by any specified integer increment. The default increment is 1.
- The counter value for the identity column is recoverable. If a failure occurs, the counter value is reconstructed from the logs, thereby guaranteeing that unique values continue to be generated.
- Identity column values can be cached to give better performance.

Ensure that Equal Values Represent the Same Entity

You can have more than one table describing the attributes 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 attributes simultaneously, you can join the two tables on the matching columns, as shown in the following example. The values 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

When you retrieve information about an entity from more than one table, ensure that 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

Normalization helps eliminate redundancies and inconsistencies in table data. It is the process of reducing tables to a set of columns where all the non-key columns depend on the primary key column. If this is not the case, the data can become inconsistent during updates.

This section briefly reviews the rules for first, second, third, and fourth normal form. The fifth normal form of a table, which is covered in many books on database design, is not described here.

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”).
Second	Each column that is not part of the key is dependent upon the key (see “Second Normal Form” on page 75).
Third	Each non-key column is independent of other non-key columns, and is dependent only upon the key (see “Third Normal Form” on page 76).
Fourth	No row contains two or more independent multi-valued facts about an entity (see “Fourth Normal Form” on page 78).

First Normal Form

A table is in *first normal form* if there is only one value, never a set of values, in each cell. A table that is in first normal form does not necessarily satisfy the criteria 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 8. 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 same table in first normal form.

Table 9. Table Conforming to First Normal Form

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY
P0010	Warehouse A	400
P0010	Warehouse B	543

Table 9. Table Conforming to First Normal Form (continued)

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY
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 part of the key is dependent upon the *entire* key.

Second normal form is violated when a non-key column is dependent upon *part* of a composite key, as in the following example:

Table 10. Table Violating Second Normal Form

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

The primary key is a composite key, consisting of the PART and the WAREHOUSE columns together. Because the WAREHOUSE_ADDRESS column 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 a warehouse changes, every row referring to a part stored in that warehouse must be updated.
- Because of this 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 a warehouse, there might not be a row in which to record the warehouse address.

The solution is to split the table into the following two tables:

Table 11. PART_STOCK Table Conforming 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 12. 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

There is a performance consideration in having the two tables in second normal form. Applications 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 is independent of other non-key columns, and is dependent only on the key.

The first table in the following example contains the columns EMPNO and WORKDEPT. Suppose a column DEPTNAME is added (see Table 14 on page 77). The new column depends on WORKDEPT, but the primary key is EMPNO. 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. Note that there are now two different department names used for department number E11. The inconsistency that results is shown in the updated version of the table.

Table 13. 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

Table 14. Unnormalized EMPLOYEE_DEPARTMENT Table After Update. Information in the 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 creating a new table, with columns for WORKDEPT and DEPTNAME. An update like changing a department name is now much easier; only the new table needs to be updated.

An SQL query that returns the department name along with the employee name is more complex to write, because it requires joining the two tables. It will probably also take longer to run than a query on a single table. Additional storage space is required, because the WORKDEPT column must appear in both tables.

The following tables are defined as a result of normalization:

Table 15. 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 16. 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 17. 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:

Table 18. EMPLOYEE_SKILL Table Conforming to Fourth Normal Form

EMPNO (Primary Key)	SKILL (Primary Key)
000130	Data Modelling
000130	Database Design
000130	Application Design

Table 19. EMPLOYEE_LANGUAGE Table Conforming to Fourth Normal Form

EMPNO (Primary Key)	LANGUAGE (Primary Key)
000130	English
000130	Spanish

If, however, the attributes are interdependent (that is, the employee applies certain languages only to certain skills), the table should *not* be split.

A good strategy when designing a database is to arrange all data in tables that are in fourth normal form, and then to decide whether the results give you an acceptable level of performance. If they do not, you can rearrange the data in tables that are in third normal form, and then reassess performance.

Planning for Constraints Enforcement

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

Unique Constraints	Ensure that key values in a table are unique. Any changes to the columns that make up the primary 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 to be taken when called by an update, delete, or insert operation on a specified table.

Unique Constraints

A *unique constraint* is the rule that ensures that key values 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 statement, 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. Moreover, 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 a unique index (if needed), and designates it as either a primary or a unique system-required index. The constraint is enforced 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

The database manager maintains referential integrity through *referential constraints*, which require that all values for a given attribute or table column also exist in some other table or column. For example, a referential constraint might require that every employee in the EMPLOYEE table 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 efficiently. When planning for referential integrity, identify all of the relationships between database tables. You can identify a relationship by defining a primary key and referential constraints.

Consider the following related tables:

Table 20. 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 21. 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

Many of the following concepts, useful for understanding referential integrity, are discussed in relation to these tables.

A *unique key* is a column or a set of columns where no values in a row are duplicated in any other row. You can define one unique key as the primary key for the table. 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 have only one primary key. In the preceding tables, DEPTNO and EMPNO are the primary keys of the DEPARTMENT and EMPLOYEE tables, respectively.

A *foreign key* is a column or a set of columns in a table that refer to a unique key or the primary key of the same or another table. A foreign key is used to establish a relationship with a unique key or the 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, DEPTNO, in the DEPARTMENT table.

A *composite key* is a key that has more than one column. Both 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 create the key for the DEPARTMENT table.

A *parent key* is a primary key or a unique key of a referential constraint. The *primary key constraint* is the default parent key of a referential constraint when a set of parent key columns is not specified.

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 row.

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 table in an arbitrary number of relationships. The EMPLOYEE table contains the foreign key WORKDEPT, and is dependent on the DEPARTMENT table, whose primary key is DEPTNO.

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 can 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 had another column called MGRID that contains the EMPNO of each employee's manager, the EMPLOYEE table would be a self-referencing table. MGRID would be a foreign key for the EMPLOYEE table.

A self-referencing table is both a parent and a dependent in the same relationship. A self-referencing row is a row that is both 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 for the foreign key in a row of a self-referencing table matches the value of the unique key in that row, the row is self-referencing.

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

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 section describes the effects of maintaining referential integrity on SQL INSERT, DELETE, UPDATE, and DROP operations.

DB2 does *not* automatically enforce referential constraints across systems. Consequently, if you want to enforce referential constraints across systems, your applications must contain the necessary logic.

The following topics are discussed:
- "INSERT Rules"
- "DELETE Rules" on page 83
- "UPDATE Rules" on page 84.

INSERT Rules: You can insert a row at any time into a parent table without any action being taken in dependent tables. However, you can only insert a row into a dependent table if 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 that 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.

If 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, none of the rows are inserted.

DELETE Rules: When you delete a row from a parent table, DB2 checks if there are any dependent rows in dependent tables with matching foreign key values. If any dependent rows are found, several actions are possible. 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 violates the referential constraint, and is not allowed.
NO ACTION	Enforces the presence of a parent row for every child after all referential constraints are applied.
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 automatically deletes the dependent rows referencing a primary key. The dependent rows do not need to be deleted first. If some of these dependent rows have dependents of their own, the delete rule for those relationships is applied. DB2 manages 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 remain unchanged.

If no delete rule is explicitly defined when the table is created, the NO ACTION rule applies.

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 for the self-referencing relationship must also be CASCADE.

You can, at any time, delete rows from a dependent table without taking any action against the parent table. In the department-employee relationship, for example, an employee could retire and have his row deleted from the employee table with no effect on the department table. (In the reverse relationship of employee-department, 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: DB2 prevents the update of a unique key for a parent row. When you update a foreign key in a dependent table, and that foreign key is not NULL, it must match some value of the parent key for 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 (except 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,
- Updating the foreign keys in dependent tables to include another valid key value.

If 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, part of the unique key may be updated. However, no more than one row can be selected for update in this case, because you are working with a unique key, and duplicate rows are not allowed.

Table Check Constraints

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

A table check constraint can be used for validation. For example, values for a department number must be within the range of 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 information about the impact of table check constraints on the IMPORT and LOAD commands.

Triggers

A *trigger* is a defined set of actions that are performed whenever a delete, insert, or update operation is carried out against a specified table. Triggers can be defined to help support business rules. Triggers can also be used to automatically update summary or audit data. Because triggers are stored in the database, 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 DB2, 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 changes, only the triggers need to be modified.

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 about triggers, see ″Creating a Trigger″ in the *Administration Guide: Implementation* and refer to the *Application Development Guide*.

Other Database Design Considerations

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

There are other issues that you may want to consider in your design, such as *audit activities*, *historical data*, *summary tables*, *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 is changed. Updates to this table could be made automatically if an appropriate trigger is defined. Audit activities can also be carried out through the DB2 audit facility. For more information, see ″Auditing DB2 Activities″ in the *Administration Guide: Implementation*.

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.

You may also want to make use of *summary* information. For example, you may have a table that has all of your employee information in it. However, you would like to have this 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. For more information about summary tables, see ″Creating a Summary Table″ in the *Administration Guide: Implementation*.

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. For more information about security and authorizations, see ″Controlling Database Access″ in the *Administration Guide: Implementation*.

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.

For more information about implementing a design that includes typed rows and tables, see "Implementing Your Design" in the *Administration Guide: Implementation*. Refer to the *Data Movement Utilities Guide and Reference* for information about 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 Enterprise - Extended 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. Each partition (or node) works in parallel to handle SQL or utility operations.

Issues and considerations relating to parallel operations are included throughout this book.

Chapter 8. Physical Database Design

After you have completed your logical database design (see "Chapter 7. Logical Database Design" on page 63), there are a number of issues you should consider about the physical environment in which your database and tables will reside. These 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 the table spaces that are required to store your data.

The following topics are covered:

- "Database Directories"
- "Estimating Space Requirements for Tables" on page 91
- "Additional Space Requirements" on page 99
- "Designing Nodegroups" on page 100
- "Designing and Choosing Table Spaces" on page 108
- "Federated Database Design Considerations" on page 129

Database Directories

When a database is created, DB2 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 devices.

The database is created in the instance that is defined by the DB2INSTANCE environment variable, or in the instance to which you have explicitly attached (using the ATTACH command). For an introduction to instances, see "Using Multiple Instances of the Database Manager" in the *Administration Guide: Implementation*.

The naming scheme used on UNIX based systems is:

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

The naming scheme used on OS/2 and the Windows operating systems:

```
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 on which 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 create the database. (On OS/2 and the Windows operating systems, the subdirectories are created under the root directory for a volume that is identified by a ″drive letter″.) These instance and database subdirectories are created within the path specified on 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 Files

The following files are associated with a 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 the *logsecond* database configuration parameters. The size of the individual files is controlled by the *logfilsiz* database configuration parameter. With circular logging, the files are reused and the same numbers remain. With archive logging, the file numbers increase in sequence as logs are archived and new logs are allocated. When 9999999 is reached, the number wraps.

By default, these log files are stored in a directory called SQLOGDIR. SQLOGDIR is found in the SQL*nnnnn* subdirectory.

SQLINSLK
This file helps to ensure that a database is used by only one instance of the database manager.

SQLTMPLK
This file helps to ensure that a database is used by only 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 backup copy of `SQLSPCS.1`. Without one of these files, you will not be able to access your database.

SQLBP.1
This file contains the definition of all buffer pools used in the database.

SQLBP.2
This file is a backup copy of `SQLBP.1`. 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 backup and restore operations.

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 and the Control Center.
2. Do not move these files.
3. Do not remove these files.
4. The only supported means of backing up a database or a table space is through the **sqlubkp** (Backup Database) API, including the command line processor and Control Center implementations of that API.

Estimating Space Requirements for Tables

Estimating the size of database objects is an imprecise undertaking. Overhead caused by disk fragmentation, free space, and the use of variable length columns makes size estimation difficult, because there is such a wide range of possibilities for column types and row lengths. After initially estimating your database size, create a test database and populate it with representative data.

From the Control Center, you can access a number of utilities that are designed to assist you in determining the size requirements of various database objects:

- You can select an object and then use the ″Estimate Size″ utility. This utility can tell you the current size of an existing object, such as a table. You can then change the object, and the utility will calculate new estimated values for the object. The utility will help you approximate storage requirements, taking future growth into account. It gives more than a single estimate of the size of the object. It also provides possible size ranges for the object: both the smallest size, based on current values, and the largest possible size.
- You can determine the relationships between objects by using the ″Show Related″ window.
- 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. For information about this utility, refer to the *Command Reference*.

In each of these cases, either the ″Show SQL″ or the ″Show Command″ button is available to you. You can also save the resulting SQL statements or commands in script files to be used later. All of these utilities have online help to assist you.

Keep these utilities in mind as you work through the planning of your physical database requirements.

When estimating the size of a database, the contribution of the following must be considered:

- “System Catalog Tables” on page 93
- “User Table Data” on page 93
- “Long Field Data” on page 95
- “Large Object (LOB) Data” on page 95
- “Index Space” on page 96

Space requirements related to the following are not discussed:

- 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

System Catalog Tables

System catalog tables are created when a database is created. The system tables grow as database objects and privileges are added to the database. Initially, they use approximately 3.5 MB of disk space.

The amount of space allocated for the catalog tables depends on the type of table space, and the extent size of 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 108.

Note: For databases with multiple partitions, the catalog tables reside only on the partition from which the CREATE DATABASE command 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 page (regardless of page size) contains 76 bytes of overhead for the database manager. This leaves 4020 bytes to hold user data (or rows), although no row on a 4 KB page 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.

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 for these columns. (See “Long Field Data” on page 95 and “Large Object (LOB) Data” on page 95 for information about estimating the space requirements for table objects that do contain these data types.)

Rows are usually inserted into a table in 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 space left on the page to contain it. If this is the case, a record is created in the original row location that points to the new location in the table file of the updated row.

If the ALTER TABLE APPEND ON statement is invoked, data is always appended, and information about any free space on the data pages is not kept. For more information about this statement, refer to the *SQL Reference*.

The number of 4 KB pages for each user table in the database can be estimated by calculating:

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

and then inserting the result into:

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

where the average row size is the sum of the average column sizes, (For information about the size of each column, refer to the CREATE TABLE statement in the *SQL Reference*.), and the factor of ″1.1″ is for overhead.

Note: This formula only provides an estimate. Accuracy of the estimate is reduced if the record length varies because of fragmentation and overflow records.

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 have a matching page size. A single table or index object can be as large as 512 GB, assuming a 32 KB 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. Maximum row lengths also vary, depending on page size:
- When the page size is 4 KB, the row length can be up to 4005 bytes.
- When the page size is 8 KB, the row length can be up to 8101 bytes.
- When the page size is 16 KB, the row length can be up to 16 293 bytes.
- When the page size is 32 KB, the row length can be up to 32 677 bytes.

Having a larger page size facilitates a reduction in the number of levels in any index. If you are working with OLTP (online transaction processing) applications, which perform 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 (decision support system) applications, which access large numbers of consecutive rows at a time, a larger page size is better, because it reduces the number of I/O requests required to read a specific number of rows. An exception 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. (There is a maximum of only 255 rows per page.) To reduce this wasted space, a smaller page size may be more appropriate.

You cannot restore a backup to a different page size.

You cannot import IXF data files that represent more than 755 columns. For more information about importing data into tables, and IXF data files, refer to the *Data Movement Utilities Guide and Reference*.

Declared temporary tables can only be created in their own ″user temporary″ table space type. There is no default user temporary table space. Temporary tables cannot have LONG data. The tables are dropped implicitly when an application disconnects from the database, and estimates of their space requirements should take this into account.

Long Field Data

Long field data is stored in a separate table object that is structured differently from other data types (see “User Table Data” on page 93 and “Large Object (LOB) Data”).

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 32 768 bytes.)

Long field data types (LONG VARCHAR or LONG VARGRAPHIC) are stored in a way 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 into the record along 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

Large object (LOB) data is stored in two separate table objects that are structured differently from other data types (see “User Table Data” on page 93 and “Long Field Data”).

To estimate the space required by LOB 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 LOB data, you can specify the COMPACT option on the *lob-options* clause of the CREATE TABLE and the ALTER TABLE statements. The COMPACT option minimizes the amount of disk space required by allowing the LOB data to be split into smaller segments. This process does not involve data compression, but simply uses the minimum amount of space, to the nearest 1 KB boundary. Using the COMPACT option may result in reduced performance when appending to LOB values.

The amount of free space contained in LOB data objects is 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 that are 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 page 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 into the record along 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. (When estimating 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, and space required for sorting during index creation.

Note: In the case of non-unique indexes, only four 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 formulas can be used to estimate the number of leaf pages (the second provides a more accurate estimate). The accuracy of these estimates depends largely on how well the averages reflect the actual data.

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

- A rough estimate of the average number of keys per leaf page is:

  ```
  (.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 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 value, if a percent free value other than the default value of ten percent was specified during index creation.
- A more accurate estimate of the average number of keys per leaf page is:

  ```
  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 sizes of all indexes on a table, and round up to a multiple of the extent size for the table space on which 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, the usable space on a page, is approximately equal to the page size minus 100. For a page size of 4096, U is 3996.
– 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 the calculation by setting the value to 0).
– M = U / (8 + *minimumKeySize* for non-leaf pages)
– K = *averageKeySize* for non-leaf pages

The *minimumKeySize* and the *averageKeySize* for non-leaf pages will be the same as for leaf pages, except when there are include columns. Include columns are not stored on non-leaf pages.

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

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

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

where:

– 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 = L / N
– Z is the number of levels in the index tree (1 initially).

Total number of pages is:

```
T = (L + P + 2) * 1.0002
```

The additional 0.02 percent is for overhead, including space map pages.

The amount of space required to create the index is estimated as:

```
T * pagesize
```

Additional Space Requirements

Additional space is also required for:
- "Log File Space"
- "Temporary Work Space" on page 100

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, defined in the database configuration file
- *logsecond* is the number of secondary log files, defined in the database configuration file
- *logfilsiz* is the number of pages in each log file, 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 about the *logprimary*, *logsecond*, and *logfilsiz* configuration parameters.

Note: The total active log space cannot exceed 32 GB.

The upper limit of log file 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 required, or may be needed only during occasional periods of high volume activity.

If the database is enabled for roll-forward recovery, special log space requirements should be taken into consideration:

- 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 sorting operations that cannot be done in memory). These temporary tables require disk space; the amount of space required is dependent upon the queries, and the size of returned tables, and cannot be estimated.

You can use the database system monitor and the query table space APIs to track 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 as little as one database partition, or as much as the entire set of database partitions defined for the database system.

You create a new nodegroup using the CREATE NODEGROUP statement, and can modify it 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 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. For more information about partitioning maps, see “Partitioning Maps” on page 103. For more information about partitioning keys, see “Partitioning Keys” on page 104.

In 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 a part of the database, complete with 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 system temporary table spaces. IBMDEFAULTGROUP is the default nodegroup for the table spaces containing the user defined tables that you may choose to put there. A user temporary table space for a declared temporary table can be created in IBMDEFAULTGROUP or any user-created nodegroup, but not in IBMTEMPGROUP.

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 containing 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-partition nodegroups, except when 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 DB2 to utilize more efficient join strategies. Collocated tables can reside in a single-partition nodegroup. Tables are considered collocated if they reside in a multiple partition nodegroup, have the same number of columns in the partitioning key, and if 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 partition nodegroup than on a 32 partition nodegroup.

You can use nodegroups to separate online transaction processing (OLTP) tables from decision support (DSS) tables, to ensure that the performance of OLTP transactions is not adversely affected.

Nodegroup Design Considerations

Your logical database design, and the amount of data to be processed, will suggest whether your database needs to be partitioned. This section covers the following topics related to database partitioning:

- “Data Partitioning”
- “Partitioning Maps” on page 103
- “Partitioning Keys” on page 104
- “Table Collocation” on page 106
- “Partition Compatibility” on page 107
- “Replicated Summary Tables” on page 107

Data Partitioning

DB2 supports a partitioned storage model that allows 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 though it were located in the same place. Applications and users accessing data in a partitioned database do not need to be aware of the physical 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 across which and how many database partitions their table data can be spread, by selecting the table space and the associated nodegroup in which the data should be stored. In addition, an updatable partitioning map 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 to having all tables divided 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).

An alternative to consider when you want tables to be positioned on each database partition, is to use summary tables and then replicate those tables. You can create a summary table containing the information that you need, and then replicate it to each node. For more information, see “Replicated Summary Tables” on page 107.

Partitioning Maps

In a partitioned database environment, the database manager must have a way of knowing which table rows are stored on which database partition. The database manager must know where to find the data it needs, and uses a map, called a *partitioning map*, to find the data.

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 27 on page 104 shows how the row with the partitioning key value (c1, c2, c3) is mapped to partition 2, which, in turn, references database partition n5.

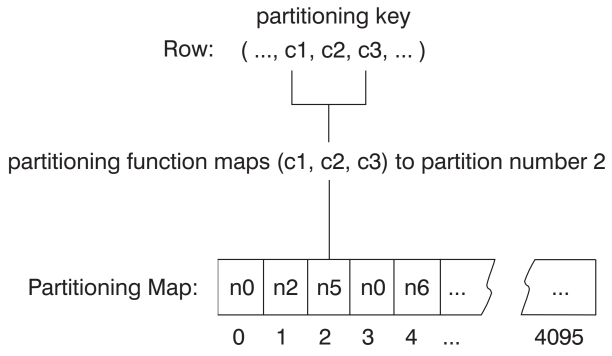

partitioning key
Row: (..., c1, c2, c3, ...)
partitioning function maps (c1, c2, c3) to partition number 2
Partitioning Map:
n0 n2 n5 n0 n6 ... ...
0 1 2 3 4 ... 4095

Figure 27. 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. This utility allows you to rebalance or introduce skew into the data distribution. For more information about this utility, refer to "Redistributing Data Across Database Partitions" in the *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 about this API, refer to the *Administrative API Reference*.

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 (LOB) data types). If you are creating a table in a table space associated with a single-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 for a default partitioning key, the table is created without one. Tables without a partitioning key are only allowed in single-partition nodegroups. You can add or drop partitioning keys at a later time, using the ALTER TABLE statement. Altering the partition key can only be done to a table whose table space is associated with a single-partition nodegroup.

Choosing a good partitioning key is important. You should take into consideration:

- 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 across 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 partitioning keys in each table have the same number of columns
- The data types of the corresponding columns are partition-compatible.

These characteristics ensure that rows of collocated tables with the same partitioning key values are located on the same partition. For more information about partition-compatibility, see "Partition Compatibility" on page 107. For more information about table collocation, see "Table Collocation" on page 106.

An inappropriate partitioning key can cause uneven data distribution. 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 result in 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 multiple partition table that contains only long data types (LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, or DBCLOB) is not supported.
- The partitioning key definition cannot be altered.
- 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 of 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 this case, the EMP_NO column would make a good single column partitioning key for EMP_TABLE.

| If the DB2_UPDATE_PART_KEY registry variable is set to NO, you cannot
| update the partitioning key column value for a row in the table and you can
| only delete or insert partitioning key column values.

Hash partitioning is the method by which 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, and generates a partition number between zero and 4095.
2. 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 partitioning map. For more information about partitioning maps, see "Partitioning Maps" on page 103.
3. The partition 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

You may discover that two or more tables frequently contribute data in response to certain queries. In this case, you will want related data from such 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" on page 107.

DB2 has the ability to recognize, when accessing more than one table for a join or a 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. For more information, refer to "Collocated Joins" in the *Administration Guide: Performance*.

Partition Compatibility

The base data types of corresponding columns of partitioning keys are compared and can be declared *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; those of non-compatible data types may not be.
- 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 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.
- Partition compatibility does not apply to LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, and BLOB data types, because 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 DB2 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.

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 based on a table that may have been created in a single-partition nodegroup, but that you want replicated across all of the database partitions in the nodegroup. To create the replicated summary table, invoke the CREATE TABLE statement with the REPLICATED keyword.

For more information about summary tables, see ″Creating a Summary Table″ in the *Administration Guide: Implementation*.

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, as well as the impact of having to update every replica, tables that are to be replicated should be small and infrequently updated.

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

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

For more information about replicated summary tables, refer to the CREATE TABLE statement in the *SQL Reference*. For more information about collocated joins, refer to ″Collocated Joins″ in the *Administration Guide: Implementation*.

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. They 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.

For information about creating or altering a table space, see ″Creating a Table Space″, or ″Altering a Table Space″ in the *Administration Guide: Implementation*.

Since table spaces reside in nodegroups, the table space selected to hold a table defines how the data for that table is distributed 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). For improved performance, each container should use a different disk. Figure 28 on page 109 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

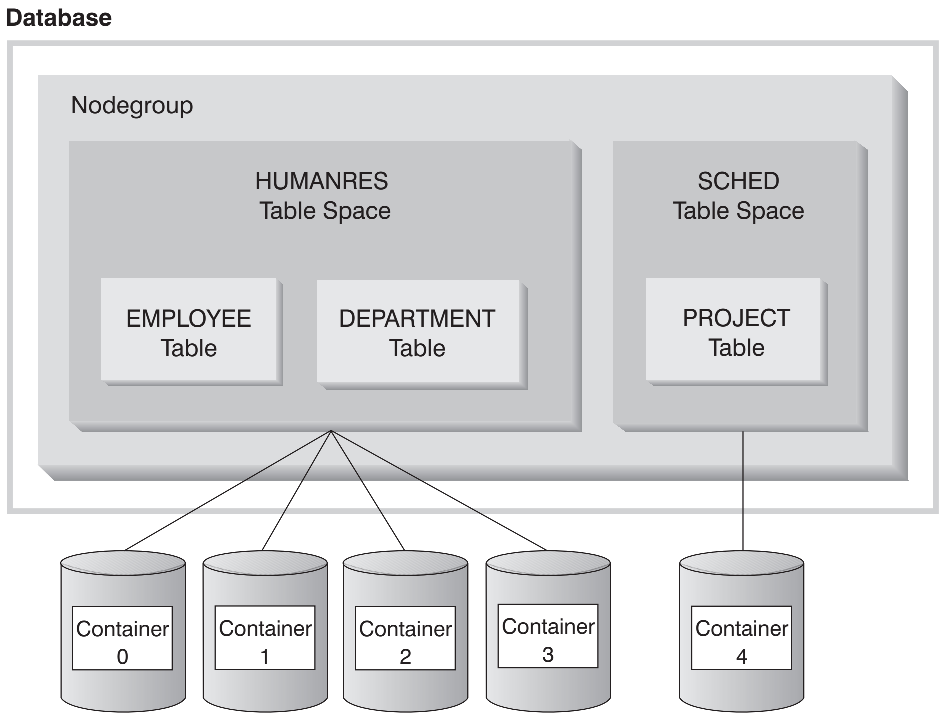


Figure 28. Table Spaces and Tables Within a Database

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 data load across containers. As a result, all containers are 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.

Figure 29 on page 110 shows the HUMANRES table space with an extent size of two 4 KB pages, and four containers, each with a small number of allocated extents. The DEPARTMENT and EMPLOYEE tables both have seven pages, and span all four containers.

HUMANRES Table Space

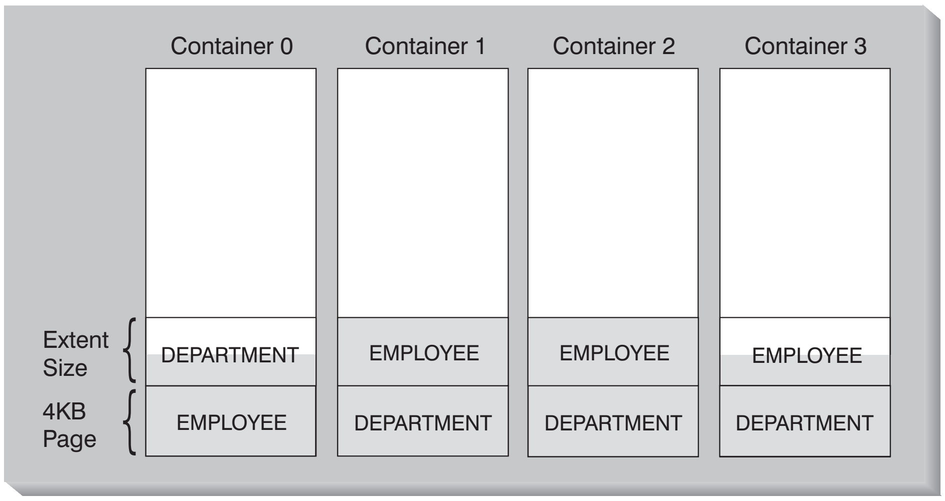


Figure 29. Containers and Extents

A database must contain at least three table spaces:

- One *catalog table space*, which contains all of 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 user-created table spaces exist, choose the one with the smallest page size large enough for this table. Otherwise, use USERSPACE1 if its page size is large enough for the table. If no table spaces with a large enough page size exist, the table is not created.

  A table's page size is determined either by row size, or the number of columns. The maximum allowable length for a row is dependent upon the page size of the table space in which the table is created. Possible values for 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, but that 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. Temporary table spaces can be *system temporary table spaces* or *user temporary*

table spaces. A database must have at least one system temporary table space; by default, one system temporary table space called TEMPSPACE1 is created at database creation time. IBMTEMPGROUP is the default nodegroup for this table space. User temporary table spaces are *not* created by default at database creation time.

| If a database uses more than one temporary table space and a new
| temporary object is needed, the optimizer will choose an approprate page
| size for this object. That object will then be allocated to the temporary table
| space with the corresponding page size. If there is more than one
| temporary table space with that page size, then the table space will be
| chosen in a round-robin fashion.

If queries are running against tables in table spaces that are defined with a page size larger than the 4 KB default (for example, an ORDER BY on 1012 columns), some of them may fail. This will occur if there are no temporary table spaces defined with a larger page size. You may need to create a temporary table space with a larger page size (8 KB, 16 KB, or 32 KB). Any DML (Data Manipulation Language) statement could fail unless there exists a temporary table space with the same page size as the largest page size in the user table space.

You should define a single SMS temporary table space with a page size equal to the page size used in the majority of your user table spaces. This should be adequate for typical environments and workloads. See also "Recommendations for Temporary Table Spaces" on page 123.

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

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

- "System Managed Space": The operating system's file manager controls the storage space.
- "Database Managed Space Table Space" on page 115: The database manager controls the storage space.

System Managed Space

In an SMS (System Managed Space) table space, the operating system's file system manager allocates and manages the space where the table is 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 distributes the data evenly across the table space containers. An SMS table space is the default table space.

Each table has at least one SMS physical file associated with it. See "SMS Physical Files" on page 114 for a list of these files and a description of their contents.

In an SMS table space, a file is extended one page at a time as the object grows. 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. In a partitioned database environment, this utility must be run on each database partition. Once multipage file allocation is enabled, it cannot be disabled. For more information about **db2empfa**, refer to the *Command Reference*.

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

- Containers for the table space.

  You must specify the number of containers that you want to use for your table space. It is very important to identify all the containers you want to use, because 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). The maximum size of the table space can be estimated by:

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

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

  Note: Care must be taken when defining the containers. If there are existing files or directories on the containers, an error (SQL0298N) is returned.

- Extent size for the table space.

  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. For more information, see "Choosing an Extent Size" on page 122.

If you do not specify the extent size when creating a table space, 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 about this parameter). This configuration parameter is initially set based on information provided when the database is created. If the *dft_extent_sz* parameter is not specified on the CREATE DATABASE command, the default extent size will be set to 32.

To choose 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.

  When you create the table space, you can specify containers that reside on different file 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 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. This process (known as *striping*) continues through the container directories until a container becomes full (SQL0289N), or no more space can be allocated from the operating system (disk full error). Striping is also used for index (`SQL`*`nnnnn`*`.INX`), long field (`SQL`*`nnnnn`*`.LF`), and LOB (`SQL`*`nnnnn`*`.LB` and `SQL`*`nnnnn`*`.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 distribute data across the containers more evenly, the database manager determines which container to use first by taking the table identifier (1 in the above example) modulo the number of containers. Containers are numbered sequentially, starting at 0.

  For more information about the files used in an SMS table space, see “SMS Physical Files” on page 114.

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 table rows are stored here, with the exception of LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, 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 `SQL`*`xxxxx`*`.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 creation or index maintenance.
SQLxxxxx.DTR	Temporary data file for the reorganization of a DAT file. When reorganizing a table, the reorg utility (through the REORG TABLE command) creates a table in one of the system 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 the reorganization of an LF file. When reorganizing a table, the reorg utility (through the REORG TABLE command) creates a table in one of the system temporary table spaces. These temporary table spaces can be defined to use containers different from those used for the user defined tables.

SQLxxxxx.RLB
Temporary data file for the reorganization of an LB file. When reorganizing a table, the reorg utility (through the REORG TABLE command) creates a table in one of the system temporary table spaces. These temporary table spaces can be defined to use containers different from those used for the user defined tables.

SQLxxxxx.RBA
Temporary data file for the reorganization of an LBA file. When reorganizing a table, the reorg utility (through the REORG TABLE command) creates a table in one of the system temporary table spaces. These temporary table spaces can be defined to use containers different from those used for the user defined tables.

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 and the Control Center.
2. Do not move these files.
3. Do not remove these files.
4. The only supported means of backing up a database or a table space is through the **sqlubkp** (Backup Database) API, including the command line processor and Control Center implementations of that API.

Database Managed Space Table Space

In a DMS (Database Managed Space) 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 those devices. The 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 that belong to the table space, and 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 regular table spaces is 64 GB for 4 KB pages; 128 GB for 8 KB pages; 256 GB for 16 KB pages; and 512 GB for 32 KB pages. The maximum size of long table spaces is 2 TB.
- Unlike SMS table spaces, the containers that make up a DMS table space do not need to be the same size; however, this is not normally recommended, because it results in uneven striping across the containers, and sub-optimal performance. If any container is full, DMS table spaces use available free space from other containers.
- Because space is pre-allocated, it 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 on it. 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 5 000 pages, and the device container is defined to allocate 3 000 pages, 2 000 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, so for optimal space management, you can use the following formula to determine an appropriate size to use when allocating a container:

  ```
  (extent_size * n) + 1
  ```

  where *extent_size* is the size of each extent in the table space, and *n* is the number of extents that you want to store in the container.
- Three extents in the table space are reserved for overhead.
- At least two extents are required to store any user table data. (These extents are required for the regular data for one table, and 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 run-time overhead associated with the file system. 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.

- If your workload involves LOBs or LONG VARCHAR data, you may derive performance benefits from file system caching. Note that 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 larger than the size allowed by the operating system.

Adding Containers to DMS Table Spaces

The ALTER TABLESPACE statement lets you add a container to an existing table space to increase its storage capacity. The contents of the table space are then rebalanced across all containers. Access to the table space is not restricted during rebalancing. 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 rebalance 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 command. Adding new containers should be done before the existing containers are almost or completely full. The new space across all containers is not available until rebalancing 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

This section covers the following topics:

- "Considerations for Table Space Input and Output (I/O)" on page 118
- "Mapping Table Spaces to Buffer Pools" on page 119
- "Mapping Table Spaces to Nodegroups" on page 120
- "Mapping Tables to Table Spaces" on page 120
- "Choosing an Extent Size" on page 122
- "Recommendations for Temporary Table Spaces" on page 123
- "Recommendations for Catalog Table Spaces" on page 124
- "Workload Considerations" on page 125
- "Choosing an SMS or DMS Table Space" on page 126
- "Optimizing Performance When Data is Placed on RAID Devices" on page 127.

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. Following are 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) are 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 or the I/O subsystem is operating at maximum capacity.

Page cleaning
As pages are read and modified, they accumulate in the database buffer pool. When a page is read in, it is read into a buffer pool page. 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 agents write out modified pages to guarantee the availability of buffer pool pages for future read requests.

Whenever it is advantageous to do so, 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 operation depends on the extent size — the bigger the extent size, the more pages can be read at one time.

How the extent is stored on disk affects I/O efficiency. In 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. If files are being used, however, 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. A large file that has been pre-allocated for use by a DMS table space tends to be contiguous on disk, especially if the file was allocated in a clean file space.

You can control the degree of prefetching 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* database configuration parameter.) The PREFETCHSIZE parameter tells DB2 how many pages to read whenever a prefetch is triggered. By setting PREFETCHSIZE to be 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* database 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, DB2 can do a big-block read from each device in parallel, thereby significantly increasing 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 run time based on query speed, buffer pool utilization, and other factors.

Some file systems use their own prefetching method (such as the Journaled File System on AIX). In some cases, file system prefetching is set to be more aggressive than DB2 prefetching. This may cause prefetching for SMS and DMS table spaces with file containers to appear to outperform prefetching for DMS table spaces with devices. This is misleading, because 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. For example, there could be a parallel prefetch request that reads three extents from a table space, and for each page being read, one modified page is written out from the buffer pool. The prefetch request may be slowed down 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 about these topics, 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. For table spaces with one or more large tables that are accessed randomly by users, the size of the buffer pool can be limited, because caching the data pages might not be beneficial.

The table space for an online transaction application might be associated with a larger buffer pool, so that the data pages used by the application can be cached longer, resulting in faster response times. Care must be taken in configuring new buffer pools. For more information on this topic, refer to "Managing the Database Buffer Pool" in the *Administration Guide: Performance*.

Note: If you have determined that a page size of 8 KB, 16 KB, or 32 KB is required by your database, 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 the database is started. If DB2 is unable to obtain the required storage, the database manager will start up with default buffer pools (one each of 4 KB, 8 KB, 16 KB, and 32 KB page sizes), and issue a warning.

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 different sizes on different partitions. For more information about the CREATE BUFFERPOOL statement, refer to the *SQL Reference*.

Mapping Table Spaces to Nodegroups

In a partitioned database environment, each table space is associated with a specific nodegroup. This allows 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. In a single-partition environment, each table space is associated with the default nodegroup. The default nodegroup, when defining a table space, is IBMDEFAULTGROUP, unless a system temporary table space is being defined; then IBMTEMPGROUP is used. For more information about the CREATE NODEGROUP statement, refer to the *SQL Reference*. For more information about nodegroups and physical database design, see “Designing Nodegroups” on page 100.

Mapping Tables to Table Spaces

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

- The partitioning of your tables.

  At a minimum, you should ensure that the table space you choose is in a nodegroup with the partitioning you want.
- 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, 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 122 for additional information.)

- The type of data in the table.

  You may, for example, have tables containing historical data that is used infrequently; 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 for which the data has to be readily available and for which you require 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 distribute your table data across three different table spaces: one for index data; one for LOB and long field data; and 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 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 those table spaces together if roll-forward recovery is enabled. SMS table spaces do not support this type of data distribution across table spaces.
- Administrative issues.

  Some administrative functions can be performed at the table space level instead of the database or table level. For example, taking a backup 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 can restore a database or a table space. If unrelated tables do not share table spaces, you have the option to restore a smaller portion of your database and reduce costs.

A good approach is to group related tables in a set of table spaces. These tables could be related through referential constraints, or through other defined business constraints.

If you need to drop and redefine a particular table often, you may want to define the table in its own table space, because 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 represents 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 one 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. This is where all indexes defined on the table are stored.

  – A long field object. This is where long field data, if your table has one or more LONG columns, is stored.

  – 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 each object allocates new extents as needed. Each table object is also paired with a meta-data object called an *extent map*, which describes all of the extents in the table space that belong to the table object. Space for extent maps is also allocated one 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 a 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 its relationship to the extent size.)

- The minimum number of extents required.

  If there is not enough space in the containers for five extents of the table space, 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 to experiment with different temporary table space configurations and workloads. The following points should be considered:

- Temporary tables are in most cases accessed in batches and sequentially. That is, a batch of rows is inserted, or a batch of sequential rows is fetched. Therefore, a larger page size typically results in better performance, because fewer logical 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 15-byte rows would be better served by 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 would not reduce the number of required I/O requests.
- 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 this 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 that there are temporary table spaces defined for each different page size used by existing tables that you may reorganize using a temporary table space.

  You can also reorganize 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. For additional information about table reorganization, refer to *Administration Guide: Performance*.

- 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 a large buffer pool, and a single 4 KB table space with a small buffer pool.

  Note: Catalog table spaces are restricted to using 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. Pre-allocation can be difficult: Temporary table spaces hold transient data that can have a very large peak storage requirement, and a much smaller 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 attempts to keep temporary table pages in memory, rather than writing them out to 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

An SMS table space is recommended for database catalogs, 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. When using a DMS table space, a small extent size (two to four pages) should be chosen; otherwise, an 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 LOBs from disk reduces performance. Since a file system usually has its own place for storing (or caching) data, using an SMS table

space, or a DMS table space built on file containers, makes avoidance of I/O possible if the LOB has previously been referenced.

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

Another factor to consider is whether 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 you can use redirected restore to enlarge an SMS table space, the use of a DMS table space facilitates the addition of new containers.

Workload Considerations

The primary type of workload being managed by DB2 in your environment can affect your choice of what table space type to use, and what page size to specify. An online transaction processing (OLTP) workload is characterized by transactions that need random access to data and that usually return small sets of data. Given that the access is random, and involves one or a few pages, prefetching is not 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 the PREFETCHSIZE parameters on the CREATE TABLESPACE statement are not important for I/O efficiency.

A query workload is characterized by transactions that need sequential or partially sequential access to data, and that usually return large sets of data. A DMS table space using multiple device containers (where each container is on a separate disk) offers the greatest potential for efficient parallel 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 for 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 to achieve I/O parallelism.

Your goal for a mixed workload is to make single I/O requests as efficient as possible for OLTP workloads, and to maximize the efficiency of parallel I/O for query workloads.

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

- For OLTP applications that perform random row read and write operations, 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 default 4 KB pages, tables are restricted to 500 columns, while the larger page sizes (8 KB, 16 KB, and 32 KB) support 1012 columns.
- The maximum 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 an SMS Table Space:

- Space is not allocated by the system until it is required.
- Creating a database requires less initial work, because 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. If you are using 8 KB pages, the table data and the index data can be as much as 128 GB. If you are using 16 KB pages, it can be as much as 256 GB. If you are using 32 KB pages, the table data and the index data can be as much as 512 GB.

- The location of the data on the disk can be controlled, if this is allowed by the operating system.
- 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.

Note: On Solaris and PTX (IBM NUMA-Q), DMS table spaces with raw devices are strongly recommended for performance-critical workloads.

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 catalog table space, and separate DMS table spaces, with multiple containers, for each table. In addition, you will probably want to store long field data and indexes on their own table spaces.

If you choose to use DMS table spaces with device containers, you must be willing to tune and administer your environment. For more information, refer to "Performance Considerations for DMS Devices" in the *Administration Guide: Performance*.

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 (see “DB2_PARALLEL_IO”) to enable parallel I/O for the table space.
- Use the DB2_STRIPED_CONTAINERS registry variable (see “DB2_STRIPED_CONTAINERS” on page 128) 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 be created so 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 the CREATE TABLESPACE or the ALTER TABLESPACE statement) will have tags taking up a full extent. Existing containers will remain unchanged.

| To stop creating containers with this attribute, reset the variable (the default
| value is NULL), and then stop and restart your instance:

```
db2set DB2_STRIPED_CONTAINERS=
db2stop
db2start
```

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

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, a federated database might need less storage space than a conventional 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 after allocating more resources to the network at the DB2 system, than to the database manager itself.

Chapter 9. Designing Distributed Databases

A transaction is commonly referred to in DB2 as a *unit of work*. A unit of work is a recoverable sequence of operations within an application process. It is used by the database manager to ensure that a database is in a consistent state. Any reading from or writing to the database is done within a unit of work.

For example, a bank transaction might involve the transfer of funds from a savings account to a checking account. After the application subtracts an amount from the savings account, the two accounts are inconsistent, and remain so until the amount is added to the checking account. When *both* steps are completed, a point of consistency is reached. The changes can be committed and 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. The ROLLBACK statement removes these changes from the database. If the application ends normally without either of these statements being explicitly issued, the unit of work is automatically committed. If it ends abnormally in the middle of a unit of work, the unit of work is automatically rolled back. Once issued, a COMMIT or a ROLLBACK cannot be stopped. With some multi-threaded applications, or some operating systems (such as Windows), if the application ends normally without either of these statements being explicitly issued, the unit of work is automatically rolled back. It is recommended that your applications always explicitly commit or roll back complete units of work. If part of a unit of work does not complete successfully, the updates are rolled back, leaving the participating 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 132
- "Using Multiple Databases in a Single Transaction" on page 133
- "Other Configuration Considerations" on page 138
- "Understanding the Two-Phase Commit Process" on page 141
- "Recovering from Problems During Two-Phase Commit" on page 144.

For information about creating applications that use distributed databases, refer to the *Application Development Guide* and the *CLI Guide and Reference*.

Using a Single Database in a Transaction

The simplest form of transaction is to read from and write to only one database within a single unit of work. This type of database access is called a *remote unit of work*.

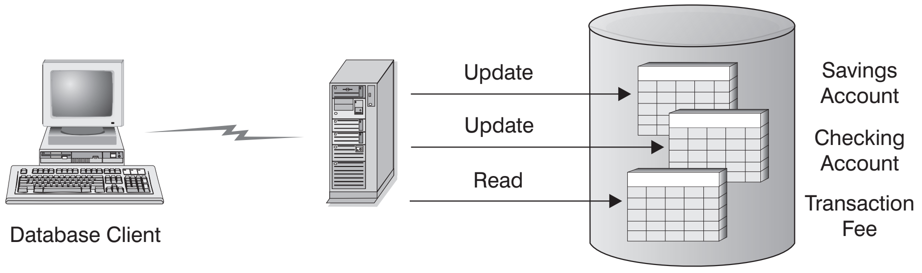


Figure 30. Using a Single Database in a Transaction

Figure 30 shows 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 must:

- Accept the amount to transfer from the user interface
- Subtract the amount from the savings account, and determine the new balance
- Read the fee schedule to determine the transaction fee for a savings account with the given balance
- Subtract the transaction fee from the savings account
- Add the amount of the transfer to the checking account
- Commit the transaction (unit of work).

To set up such an application, you must:

1. Create the tables for the savings account, checking account and banking fee schedule in the same database (see "Implementing Your Design" in the *Administration Guide: Implementation*)
2. If physically remote, set up the database server to use the appropriate communications protocol, as described in the *Installation and Configuration Supplement*
3. If physically remote, catalog the node and the database to identify the database on the database server, as described in the *Quick Beginnings* books
4. Precompile your application program to specify a type 1 connection; that is, specify CONNECT 1 (the default) on the PRECOMPILE PROGRAM command, as described in the *Application Development Guide*.

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.

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).

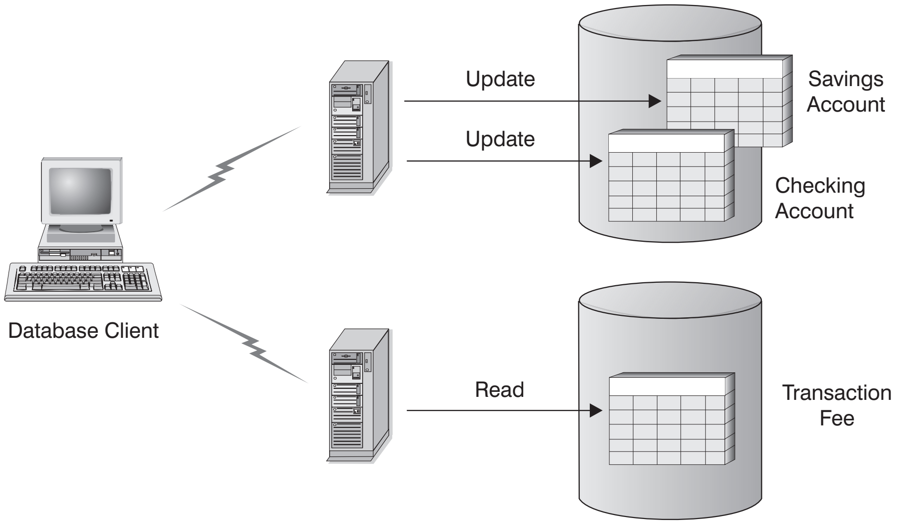


Figure 31. Using Multiple Databases in a Single Transaction

Figure 31 shows 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 one shown in Figure 30 on page 132, except for the number of databases, and the location of the tables.

To set up a funds transfer application for this environment, you must:

1. Create the necessary tables in the appropriate databases (see "Implementing Your Design" in the *Administration Guide: Implementation*)
2. If physically remote, set up the database servers to use the appropriate communications protocols, as described in the *Installation and Configuration Supplement*
3. If physically remote, catalog the nodes and the databases to identify the databases on the database servers, as described in the *Quick Beginnings* books

4. Precompile your application program to specify a type 2 connection (that is, specify CONNECT 2 on the PRECOMPILE PROGRAM command), and one-phase commit (that is, specify SYNCPOINT ONEPHASE on the PRECOMPILE PROGRAM command), as described in the *Application Development Guide*.

If databases are located on a host or AS/400 database server, you require DB2 Connect for connectivity to these servers. For information about setup, refer to one of the DB2 Connect *Quick Beginnings* books. For information about using DB2 Connect, refer to the *DB2 Connect User's Guide*.

Updating Multiple Databases

If your data is distributed across multiple databases, you may want to read and update several databases in a single transaction. This type of database access is called a *multisite update*.

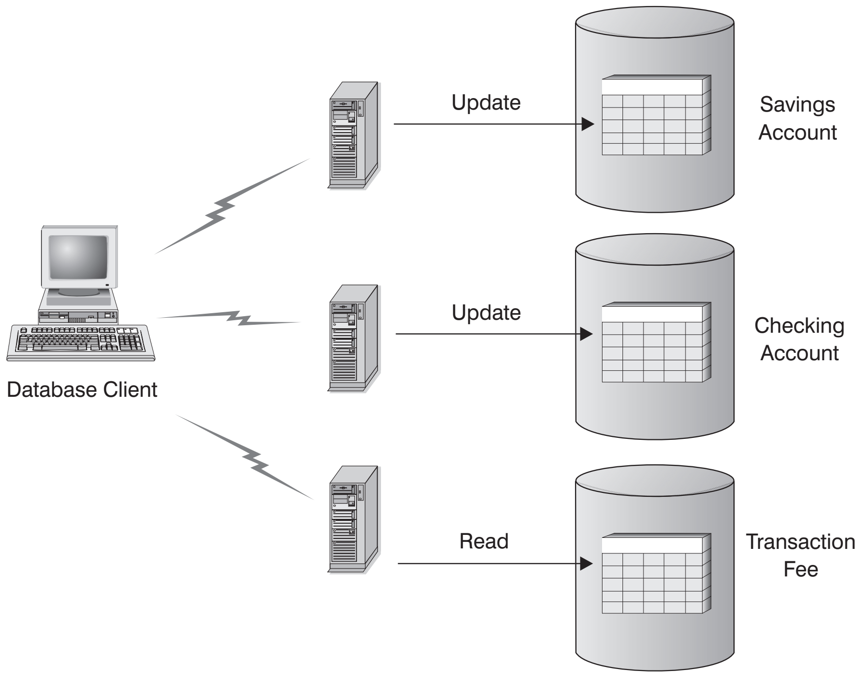


Figure 32. Updating Multiple Databases in a Single Transaction

Figure 32 shows a database client running a funds transfer application that accesses three database servers: one containing the checking account, another containing the savings account, and the third containing the banking fee schedule.

To set up a funds transfer application for this environment, you have two options:

1. With a transaction manager (TM):
   a. Create the necessary tables in the appropriate databases (see ″Implementing Your Design″ in the *Administration Guide: Implementation*).
   b. If physically remote, set up the database servers to use the appropriate communications protocols, as described in the *Installation and Configuration Supplement*.
   c. If physically remote, catalog the nodes and the databases to identify the databases on the database servers, as described in the *Quick Beginnings* books.
   d. Precompile your application program to specify a type 2 connection (that is, specify CONNECT 2 on the PRECOMPILE PROGRAM command), and two-phase commit (that is, specify SYNCPOINT TWOPHASE on the PRECOMPILE PROGRAM command), as described in the *Application Development Guide*.
   e. Configure the DB2 transaction manager (TM), as described in "Using the DB2 Transaction Manager".
2. Without a transaction manager:
   a. Create the necessary tables in the appropriate databases (see ″Implementing Your Design″ in the *Administration Guide: Implementation*).
   b. If physically remote, set up the database servers to use the appropriate communications protocols, as described in the *Installation and Configuration Supplement*.
   c. If physically remote, catalog the nodes and the databases to identify the databases on the database servers, as described in the *Quick Beginnings* books.
   d. Precompile your application program to specify a type 2 connection (that is, specify CONNECT 2 on the PRECOMPILE PROGRAM command), and one-phase commit (that is, specify SYNCPOINT ONEPHASE on the PRECOMPILE PROGRAM command), as described in the *Application Development Guide*.

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 and track its completion status.

If you are using an XA-compliant transaction manager, such as IBM TXSeries, BEA Tuxedo, or Microsoft Transaction Server, see “Chapter 10. Designing for Transaction Managers” on page 147 for integration instructions.

When using DB2 UDB for UNIX based systems, Windows operating systems, or OS/2 to coordinate your transactions, you must fulfill certain configuration requirements. If you use TCP/IP exclusively for communications, and DB2 UDB and DB2 for OS/390 are the only database servers involved in your transactions, configuration is straightforward.

DB2 UDB and DB2 for OS/390 Using TCP/IP Connectivity: If each of the following statements is true for your environment, the configuration steps for multisite update are straightforward.

- All communications with remote database servers (including DB2 UDB for OS/390) use TCP/IP exclusively.
- DB2 UDB for UNIX based systems, Windows operating systems, OS/2, or OS/390 are the only database servers involved in the transaction.
- The DB2 Connect sync point manager (SPM) is not configured.

  The DB2 Connect sync point manager is configured automatically at DB2 instance creation time, and is required when:
  - SNA connectivity is used with host or AS/400 database servers for multisite 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 10. Designing for Transaction Managers” on page 147. If your environment does not require the DB2 Connect sync point manager, you can turn it off by issuing the command `db2 update dbm cfg using spm_name NULL` at the DB2 Connect server. Then stop and restart DB2.

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*. For more information about this configuration parameter, see "Configuring DB2" in the *Administration Guide: Performance*. Consider the following factors when setting this configuration parameter:

- The transaction manager database can be:
  - A DB2 UDB for UNIX based systems, Windows operating systems, or OS/2 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, reducing the possibility of accidental power downs, reboots, and so on. Therefore the recovery logs, used in the event of resynchronization, are more secure.

- If a value of 1ST_CONN is specified for the *tm_database* configuration parameter, the first database to which an application connects is 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; that is, if:
  - 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, in your environment:

- TCP/IP is not used exclusively for communications with remote database servers (for example, NETBIOS is used)
- DB2 for MVS Version 3 or Version 4, DB2 for AS/400, or DB2 for VM&VSE is accessed
- DB2 for OS/390 is accessed using SNA
- The DB2 Connect sync point manager is used to access host or AS/400 database servers

the configuration steps for multisite update are more involved.

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*. For more information about this configuration parameter, see "Configuring DB2" in the *Administration Guide: Performance*. Consider the following factors when setting this configuration parameter:

- The transaction manager database can be a DB2 UDB for UNIX based systems, Windows operating systems, or OS/2 database.
- If a value of 1ST_CONN is specified for the *tm_database* configuration parameter, the first database to which an application connects is 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; that is, if:

- 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 Configuration Considerations

You should consider the following configuration parameters when you are setting up your environment. For additional information about setting these parameters, refer to the *DB2 Connect User's Guide*.

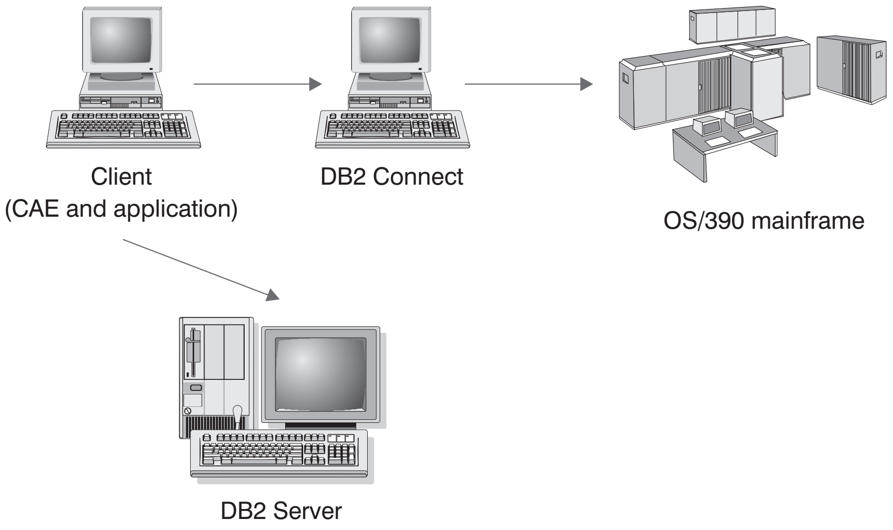


Figure 33. Configuration Considerations

Database Manager Configuration Parameters

- *tm_database*

  This parameter identifies the name of the Transaction Manager (TM) database for each DB2 instance.
- *spm_name*

  This parameter identifies the name of the DB2 Connect sync point manager instance to the database manager. For resynchronization to be successful, the name must be unique across your network.

- *resync_interval*

  This parameter identifies the time interval (in seconds) after which the DB2 Transaction Manager, the DB2 server database manager, and the DB2 Connect sync point manager or the DB2 UDB sync point manager should retry the recovery of any outstanding indoubt transactions.
- *spm_log_file_sz*

  This parameter specifies the size (in 4 KB pages) of the SPM log file.
- *spm_max_resync*

  This parameter identifies the number of agents that can simultaneously perform resynchronization operations.
- *spm_log_path*

  This parameter identifies the log path for the SPM log files.

Database Configuration Parameters

- *maxappls*

  This parameter specifies the maximum permitted number of active applications. Its value must be equal to or greater than the sum of the connected applications, plus the number of these applications that may be concurrently in the process of completing a two-phase commit or rollback, plus the anticipated number of indoubt transactions that might exist at any one time. For more information about indoubt transactions, see "Recovering from Problems During Two-Phase Commit" on page 144.
- *autorestart*

  This database configuration parameter specifies whether the RESTART DATABASE routine will be invoked automatically when needed. The default value is `YES` (that is, enabled).

  A database containing indoubt transactions requires a restart database operation to start up. If *autorestart* is not enabled when the last connection to the database is dropped, the next connection will fail and require an explicit RESTART DATABASE invocation. This condition will exist until the indoubt transactions have been removed, either by the transaction manager's resynchronization operation, or through a heuristic operation initiated by the administrator. When the RESTART DATABASE command is issued, a message is returned if there are any indoubt transactions in the database. The administrator can then use the LIST INDOUBT TRANSACTIONS command and other command line processor commands to find get information about those indoubt transactions.

For more information about these configuration parameters, refer to the *Administration Guide: Performance*.

Host or AS/400 Applications Accessing a LAN Based DB2 Universal Database Server in a Multisite Update

DB2 Universal Database does not support multisite update from the host or AS/400 database clients using TCP/IP connectivity. In this situation, only SNA (Systems Network Architecture) connectivity is supported. The DB2 sync point manager is required for multisite update. DB2 Connect is not used in this scenario.

The database server that is being accessed from the host or the AS/400 database client does not have to be local to the workstation with the DB2 sync point manager. The host or AS/400 database client could connect to a DB2 UDB server using the DB2 sync point manager workstation as an interim gateway. This allows you to isolate the DB2 sync point manager workstation in a secure environment, while the actual DB2 UDB servers are remote in your organization. This also permits a DB2 common server Version 2 database to be involved in multisite updates originating from the host or AS/400 database clients.

The steps are as follows:

- On the workstation that will be directly accessed by the host or AS/400 application:
  1. Install DB2 Universal Database Enterprise Edition, or Enterprise - Extended Edition to provide multisite update support with the 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. When the supported IBM SNA products are used, the SNA profiles required for the DB2 sync point manager are created automatically, based on the value of the *spm_name* database manager configuration parameter. Any other supported SNA stack will require manual configuration. For details, refer to the *Installation and Configuration Supplement*.
  6. Determine the value to be specified for the *spm_name* database manager configuration parameter. This parameter is pre-configured at DB2 instance creation time with a derivative of the TCP/IP host name for the machine. If this is acceptable and unique within your environment, do not change it.

7. If necessary, update *spm_name* on the DB2 Universal Database server, using the UPDATE DATABASE MANAGER CONFIGURATION command.
8. Configure communications required for this DB2 workstation to connect to remote DB2 UDB servers, if any.
9. Configure communications required for remote DB2 UDB servers to connect to this DB2 server.
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 UDB workstation.

- On each remote DB2 UDB server that will be accessed by the host or AS/400 database client:
  1. Configure communications required for the remote DB2 UDB workstation with the DB2 sync point manager to connect to this DB2 UDB server.
  2. Stop and restart the database manager.

Understanding the Two-Phase Commit Process

Figure 34 on page 142 illustrates the steps involved in a multisite update. Understanding how a transaction is managed will help you to resolve the problem if an error occurs during the two-phase commit process.

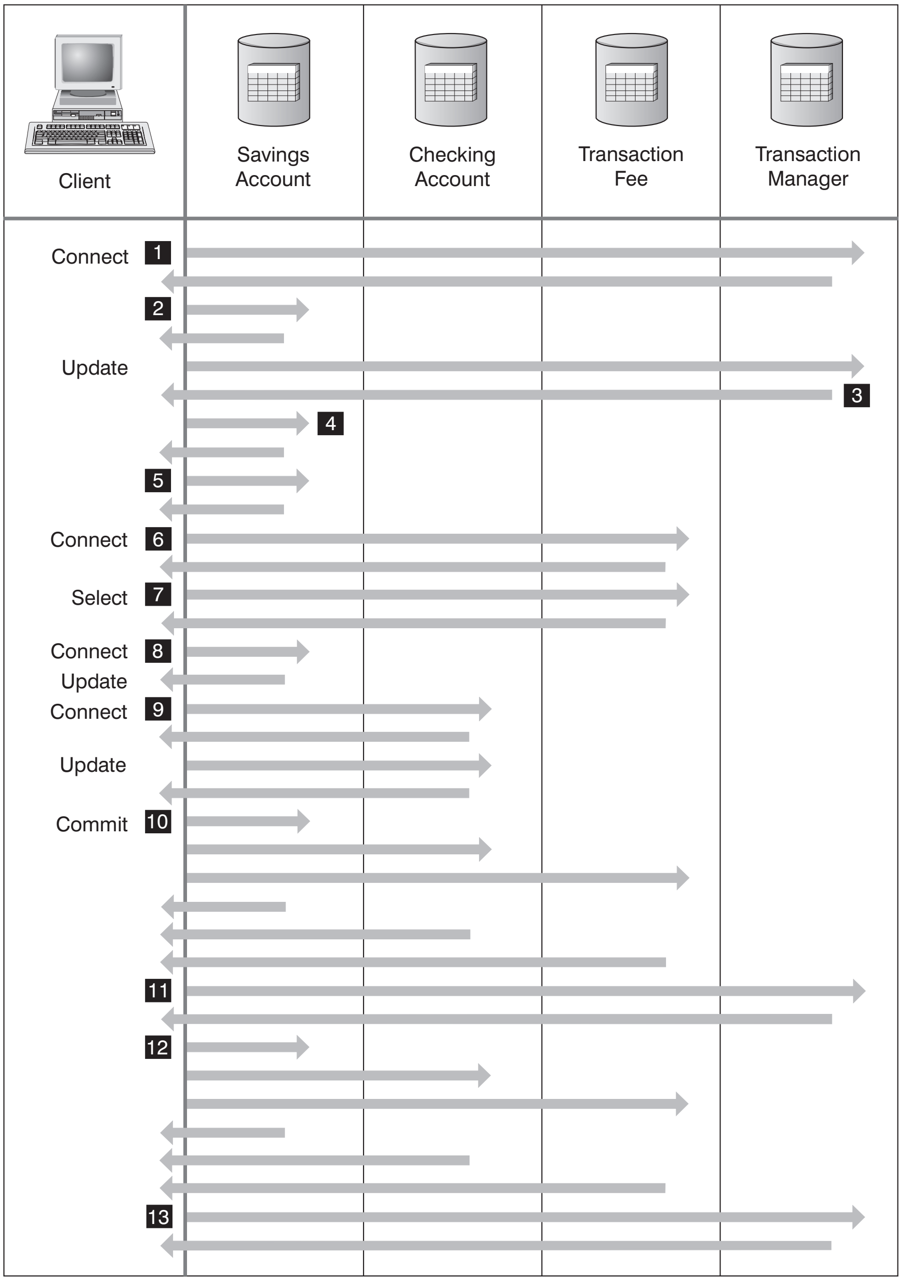


Figure 34. Updating Multiple Databases

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 DB2 CLI (Call Level Interface) configuration (refer to the *CLI Guide and Reference* for details).

1 When the database client wants to connect to the SAVINGS_DB database, it first internally connects to the transaction manager (TM) database. The TM database returns an acknowledgment to the database client. If the database manager configuration parameter *tm_database* is set to `1ST_CONN`, SAVINGS_DB becomes the transaction manager database for the duration of this application instance.

2 The connection to the SAVINGS_DB database 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 during the establishment of a connection.

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 database 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 in 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 way.

8 Additional SQL statements can be run against the SAVINGS_DB database by setting the connection, as appropriate. Since the unit of work has already been registered with the SAVINGS_DB database **4**, the database client does not need to perform the registration step again.

9 Connecting to, and using the CHECKING_DB database follows the same rules described in **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 its 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, informing 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 the client.

13 After the database client receives a positive response from all participating databases, it sends a message to the transaction manager database, informing it that the unit of work has been completed. The transaction manager database then writes a ″COMMITTED″ record to its log file, indicating that the unit of work is complete, and replies to the client, indicating that it has finished.

Recovering from Problems During Two-Phase Commit

Recovering from error conditions 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 resulting from network or communications 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 141 (points **10**, **11**, and **12**). The following explains how the database manager handles errors during the two-phase commit process:

- **First Phase Error**

  If a database communicates that it has 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 during the first phase. Each database then writes an ″ABORT″ record to its log file, and releases the locks that were held for this unit of work.

- **Second Phase Error**

  Error handling at this stage is dependent upon 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 application, however, will be informed that the commit was successful through the SQLCA. DB2 will ensure that the uncommitted transaction in the database server is committed. The database manager configuration parameter *resync_interval* (see "Configuring DB2" in the *Administration Guide: Performance*) is used to specify how long the transaction manager database should wait between attempts to commit the unit of work. All locks are held at the database server until the unit of work is committed.

  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, but have not completed the second phase of the commit process. The database manager associated with the transaction manager database performs the resynchronization by:

  1. Connecting to the databases that indicated they were "PREPARED" to commit during the first phase of the commit process.
  2. Attempting to commit the indoubt transactions at those databases. (If the indoubt transactions cannot be found, the database manager assumes that the database successfully committed the transactions during the second phase of the commit process.)
  3. Committing the indoubt transactions 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 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 that the transaction was rolled back, and will roll back the indoubt transaction in this database. Otherwise, the database waits for a commit request from the transaction manager database.

  If the transaction was coordinated by a transaction processing monitor (XA-compliant transaction manager), the database will always depend on the TP monitor to initiate the resynchronization.

If, for some reason, you cannot wait for the transaction manager to automatically resolve indoubt transactions, there are actions you can take to manually resolve them. This manual process is sometimes referred to as ″making a heuristic decision″. For more information about manual recovery of indoubt transactions, see “Making a Heuristic Decision” on page 158.

Resynchronizing Indoubt Transactions if AUTORESTART=OFF

For configuration considerations in the DB2 Universal Database two-phase commit environment, see “Other Configuration Considerations” on page 138.

In particular, if the *autorestart* database configuration parameter is set to `OFF`, and there are indoubt transactions in either the TM or RM databases, the RESTART DATABASE command is required to start the resynchronization process. When 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 invocation will be dropped, and must be restarted once again. Issue the TERMINATE command to drop the connection after no more indoubt transactions are returned by the LIST INDOUBT TRANSACTIONS command.

Chapter 10. Designing for Transaction Managers

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, described in "Updating Multiple Databases" on page 134.

The following topics will 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:

- "X/Open Distributed Transaction Processing Model" on page 148
- "Setting Up a Database as a Resource Manager" on page 152
- "xa_open and xa_close Strings Usage" on page 152
- "New xa_open String Format for DB2 Version 7" on page 152
- "TPM and TP_MON_NAME Values" on page 154
- "xa_open String Format for Earlier Versions of DB2" on page 157
- "Updating Host or AS/400 Database Servers" on page 157
- "Database Connection Considerations" on page 158
- "Making a Heuristic Decision" on page 158
- "Security Considerations" on page 161
- "Configuration Considerations" on page 162
- "XA Function Supported" on page 163
- "XA Interface Problem Determination" on page 165
- "Configuring XA Transaction Managers to Use DB2 UDB" on page 166
- "Configuring Microsoft Transaction Server" on page 171.

If you are using an XA-compliant transaction manager, or are implementing one, more information is available from our technical support web site:

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

Once there, choose "DB2 Universal Database", then search the web site using the keyword "XA" for the latest available information on XA-compliant transaction managers.

X/Open Distributed Transaction Processing Model

The X/Open Distributed Transaction Processing (DTP) model includes three interrelated components:

- "Application Program (AP)"
- "Transaction Manager (TM)" on page 150
- "Resource Managers (RM)" on page 151.

Figure 35 illustrates this model, and shows the relationship among these components.

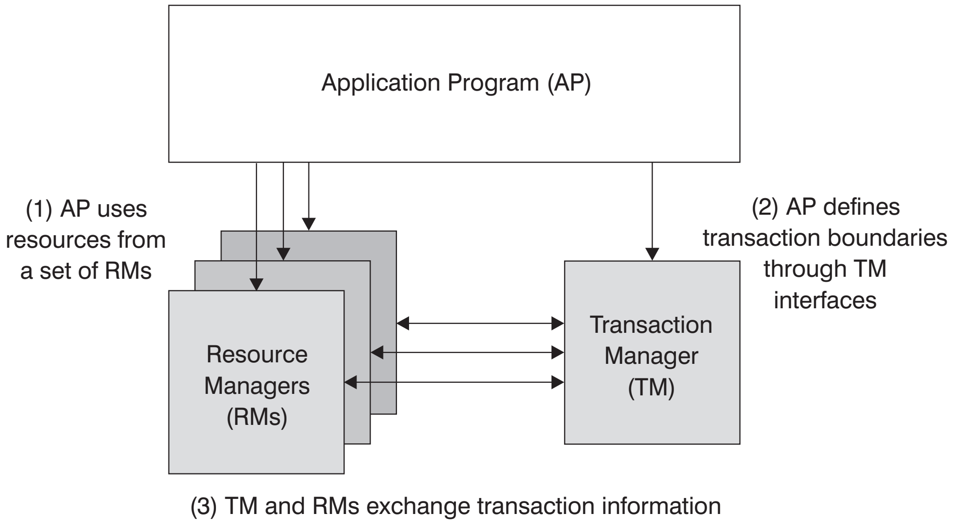

Application Program (AP)
(1) AP uses resources from a set of RMs
(2) AP defines transaction boundaries through TM interfaces
Resource Managers (RMs)
Transaction Manager (TM)
(3) TM and RMs exchange transaction information

Figure 35. X/Open Distributed Transaction Processing (DTP) Model

Application Program (AP)

The application program (AP) defines transaction boundaries, and defines 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 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 (TM) product usually includes a transaction processing (TP) monitor to run the user 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. The application program in a distributed transaction processing (DTP) environment is really a combination of 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 conserves 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 that the server processes are the actual ″user processes″ to the TM and the RMs. This has implications for security administration and application programming. For details, see “Security Considerations” on page 161.

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 TP monitor simply 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 that supports 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 loosely coupled global transactions. See “Updating Multiple Databases” on page 134.)

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 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 a *resync*) on system startup, or to let the administrator perform a *heuristic* operation (also known as *manual intervention*), if necessary.

After a TP monitor is started, it asks the TM to open all the RMs that a set of application servers have defined. The TM passes **xa_open** calls to the RMs, so that they can be initialized for DTP processing. As part of this startup procedure, the TM performs a resync to recover all *indoubt transactions*. An indoubt transaction is a global transaction that was left in an uncertain state. This occurs when 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 roll back its branch of the transaction until the TM can reconcile its own log with the RM logs when they become available again. To perform the resync operation, the TM issues a **xa_recover** call one or more times to each of the RMs to identify all the indoubt transactions. The TM compares 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 has already committed or rolled back its branch of an indoubt transaction through a heuristic operation by its administrator, the TM issues an **xa_forget** call to that RM to complete the resync operation.

When a user application requests a commit or a 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 XA TM (implemented in the Encina Server) will in turn issue XA calls, such as **xa_end**, **xa_prepare**, **xa_commit**, or **xa_rollback** to request the RM to commit or roll back 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 resource manager of a database, 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 about XA functions supported by the database manager, see "XA Function Supported" on page 163.

There are two methods by which the RM can register its participation in each global transaction: *static registration* and *dynamic registration*:

- Static registration requires the TM to issue (for every transaction) the **xa_start**, **xa_end**, and **xa_prepare** series of calls to all the RMs defined for the server application, regardless of whether a given RM is used by the transaction. This is inefficient if not every RM is involved in every transaction, and the degree of inefficiency is proportional to the number of defined RMs.
- Dynamic registration (used by DB2) is flexible and efficient. An RM registers with the TM using an **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 the **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 restrictions that the DTP software components impose. For information about X/Open XA interface programming considerations, refer to the *Application Development Guide*.

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. Integration information regarding the most popular TP monitors is provided in "Configuring XA Transaction Managers to Use DB2 UDB" on page 166.

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. For a description of DB2's xa_open string format, see "xa_open and xa_close Strings Usage".

xa_open and xa_close Strings Usage

The database manager xa_open string has two accepted formats. One format is new to DB2 Version 7. The second format is used by earlier versions of DB2, and remains for back-level compatibility. New implementations should use the new format, and older implementations should be migrated to the new format when possible. Future versions of DB2 may not support the older xa_open string format. For information about the original xa_open string format, see "xa_open String Format for Earlier Versions of DB2" on page 157.

When setting up a database as a resource manager, you do not need the xa_close string. If provided, this string will be ignored by the database manager.

New xa_open String Format for DB2 Version 7

The following xa_open string format is new to DB2 Version 7:

```
parm_id1 = <parm value>,parm_id2 = <parm value>, ...
```

It does not matter in what order these parameters are specified. Valid values for *parm_id* are described in the following table.

Table 22. Valid Values for parm_id

Parameter Name	Value	Mandatory?	Case Sensitive?	Default Value
DB	Database alias	Yes	No	None
Database alias used by the application to access the database.				
UID	User ID	No	Yes	None
User ID that has authority to connect to the database. Required if a password is specified.				
PWD	Password	No	Yes	None
A password that is associated with the user ID. Required if a user ID is specified.				
TPM	Transaction processing monitor name	No	No	None
Name of the TP monitor being used. For supported values, see "TPM and TP_MON_NAME Values" on page 154. This parameter can be specified to allow multiple TP monitors to use a single DB2 instance. The specified value will override the value specified in the *tp_mon_name* database manager configuration parameter.				

Table 22. Valid Values for parm_id (continued)

Parameter Name	Value	Mandatory?	Case Sensitive?	Default Value
AXLIB	Library that contains the TP monitor's **ax_reg** and **ax_unreg** functions.	No	Yes	None
This value is used by DB2 to obtain the addresses of the required **ax_reg** and **ax_unreg** functions. It can be used to override assumed values based on the TPM parameter, or it can be used by TP monitors that do not appear on the list for TPM.				
CHAIN_END	xa_end chaining flag. Valid values are T, F, or no value.	No	No	F
XA_END chaining is an optimization that can be used by DB2 to reduce network flows. If the TP monitor environment is such that it can be guaranteed that **xa_prepare** will be invoked within the same thread or process immediately following the call to **xa_end**, and if CHAIN_END is on, the xa_end flag will be chained with the **xa_prepare** command, thus elimintaing one network flow. A value of T means that CHAIN_END is on; a value of F means that CHAIN_END is off; no specified value means that CHAIN_END is on. This parameter can be used to override the setting derived from a specified TPM value.				
SUSPEND_ CURSOR	Specifies whether cursors are to be kept when a transaction thread of control is suspended. Valid values are T, F, or no value.	No	No	F
TP monitors that suspend a transaction branch can reuse the suspended thread or process for other transactions. In these situations, cursors must be closed so that the new transaction does not inherit them. When the suspended transaction is resumed, the application must obtain the cursors again. If SUSPEND_CURSOR is on, any open cursors are not closed, but the thread or process cannot be reused for other transactions. Only the resumption of the suspended transaction is permitted. A value of T means that SUSPEND_CURSOR is on; a value of F means that SUSPEND_CURSOR is off; no specified value means that SUSPEND_CURSOR is on. This parameter can be used to override the setting derived from a specified TPM value.				

Table 22. Valid Values for parm_id (continued)

Parameter Name	Value	Mandatory?	Case Sensitive?	Default Value
HOLD_CURSOR	Specifies whether cursors are held across transaction commits. Valid values are `T`, `F`, or no value.	No	No	F
TP monitors typically reuse threads or processes for multiple applications. To ensure that a newly loaded application does not inherit cursors opened by a previous application, cursors are closed after a commit. If HOLD_CURSOR is on, cursors are held across transaction commits. A value of `T` means that HOLD_CURSOR is on; a value of `F` means that HOLD_CURSOR is off; no specified value means that HOLD_CURSOR is on. This parameter can be used to override the setting derived from a specified TPM value.				

TPM and TP_MON_NAME Values

The xa_open string TPM parameter and the *tp_mon_name* database manager configuration parameter are used to indicate to DB2 which TP monitor is being used. The *tp_mon_name* value applies to the entire DB2 instance. The TPM parameter applies only to the specific XA resource manager. The TPM value overrides the *tp_mon_name* parameter. Valid values for the TPM and *tp_mon_name* parameters are as follows:

Table 23. Valid Values for TPM and tp_mon_name

TPM Value	TP Monitor Product	Internal Settings

CICS	IBM TxSeries CICS	`AXLIB=libEncServer (for Windows)`
		`     =/usr/lpp/encina/lib/libEncServer`
		`        (for UNIX based systems)`
		`HOLD_CURSOR=T`
		`CHAIN_END=T`
		`SUSPEND_CURSOR=F`

ENCINA	IBM TxSeries Encina Monitor	`AXLIB=libEncServer (for Windows)`
		`     =/usr/lpp/encina/lib/libEncServer`
		`        (for UNIX based systems)`
		`HOLD_CURSOR=F`
		`CHAIN_END=T`
		`SUSPEND_CURSOR=F`

Table 23. Valid Values for TPM and tp_mon_name (continued)

TPM Value	TP Monitor Product	Internal Settings

MQ	IBM MQSeries	`AXLIB=mqmax (for Windows)`
		`     =/usr/mqm/lib/libmqmax.a`
		`        (for AIX)`
		`     =/opt/mqm/lib/libmqmax.a`
		`        (for Solaris)`
		`HOLD_CURSOR=F`
		`CHAIN_END=F`
		`SUSPEND_CURSOR=F`

CB	IBM Component Broker	`AXLIB=somtrx1i (for Windows)`
		`     =libsomtrx1`
		`        (for UNIX based systems)`
		`HOLD_CURSOR=F`
		`CHAIN_END=T`
		`SUSPEND_CURSOR=F`

SF	IBM San Francisco	`AXLIB=ibmsfDB2`
		`HOLD_CURSOR=F`
		`CHAIN_END=T`
		`SUSPEND_CURSOR=F`

TUXEDO	BEA Tuxedo	`AXLIB=libtux`
		`HOLD_CURSOR=F`
		`CHAIN_END=F`
		`SUSPEND_CURSOR=F`

MTS	Microsoft Transaction Server	It is not necessary to configure DB2 for MTS. MTS is automatically detected by DB2's ODBC driver.
JTA	Java Transaction API	It is not necessary to configure DB2 for Enterprise Java Servers (EJS) such as IBM WebSphere. DB2's JDBC driver automatically detects this environment.

Examples

1. You are using IBM TxSeries CICS on Windows NT. The TxSeries documentation indicates that you need to configure *tp_mon_name* with a value of `libEncServer:C`. This is still an acceptable format; however, with DB2 UDB or DB2 Connect Version 7, you have the option of:

- Specifying a *tp_mon_name* of `CICS` (recommended for this scenario):

  ```
  db2 update dbm cfg using tp_mon_name CICS
  ```

  For each database defined to CICS in the Region—> Resources—> Product—> XAD—> Resource manager initialization string, specify:

  `db=dbalias,uid=`*`userid`*`,pwd=`*`password`*
- For each database defined to CICS in the Region—> Resources—> Product—> XAD—> Resource manager initialization string, specify:

  `db=dbalias,uid=`*`userid`*`,pwd=`*`password`*`,tpm=cics`
2. You are using IBM MQSeries on Windows NT. The MQSeries documentation indicates that you need to configure *tp_mon_name* with a value of `mqmax`. This is still an acceptable format; however, with DB2 UDB or DB2 Connect Version 7, you have the option of:
   - Specifying a *tp_mon_name* of `MQ` (recommended for this scenario):

     ```
     db2 update dbm cfg using tp_mon_name MQ
     ```

     For each database defined to CICS in the Region—> Resources—> Product—> XAD—> Resource manager initialization string, specify:

     `uid=`*`userid`*`,db=`*`dbalias`*`,pwd=`*`password`*
   - For each database defined to CICS in the Region—> Resources—> Product—> XAD—> Resource manager initialization string, specify:

     `uid=`*`userid`*`,db=`*`dbalias`*`,pwd=`*`password`*`,tpm=mq`
3. You are using both IBM TxSeries CICS and IBM MQSeries on WIndows NT. A single DB2 instance is being used. In this scenario, you would configure as follows:
   a. For each database defined to CICS in the Region—> Resources—> Product—> XAD—> Resource manager initialization string, specify:

      `pwd=`*`password`*`,uid=`*`userid`*`,tpm=cics,db=`*`dbalias`*

   b. For each database defined as a resource in the queue manager properties, specify an XaOpenString as:

      `db=`*`dbalias`*`,uid=`*`userid`*`,pwd=`*`password`*`,tpm=mq`
4. You are developing your own XA-compliant transaction manager (XA TM) on Windows NT, and you want to tell DB2 that library ″myaxlib″ has the required functions **ax_reg** and **ax_unreg**. Library ″myaxlib″ is in a directory specified in the PATH statement. You have the option of:
   - Specifying a *tp_mon_name* of `myaxlib`:

     ```
     db2 update dbm cfg using tp_mon_name myaxlib
     ```

     and, for each database defined to the XA TM, specifying an xa_open string:

     `db=`*`dbalias`*`,uid=`*`userid`*`,pwd=`*`password`*
   - For each database defined to the XA TM, specifying an xa_open string:

```
db=dbalias,uid=userid,pwd=password,axlib=myaxlib
```

5. You are developing your own XA-compliant transaction manager (XA TM) on Windows NT, and you want to tell DB2 that library ″myaxlib″ has the required functions **ax_reg** and **ax_unreg**. Library ″myaxlib″ is in a directory specified in the PATH statement. You also want to enable XA END chaining. You have the option of:
   - For each database defined to the XA TM, specifying an xa_open string:
     ```
     db=dbalias,uid=userid,pwd=password,axlib=myaxlib,chain_end=T
     ```
   - For each database defined to the XA TM, specifying an xa_open string:
     ```
     db=dbalias,uid=userid,pwd=password,axlib=myaxlib,chain_end
     ```

xa_open String Format for Earlier Versions of DB2

Earlier versions of DB2 used the xa_open string format described here. This format is still supported for compatibility reasons. Applications should be migrated to the new format (see “New xa_open String Format for DB2 Version 7” on page 152) when possible.

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 alias name of the database. The alias name is the same as the database name unless you have explicitly cataloged an alias name after you created the database. The user name 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 need the xa_close string. If provided, this string will be ignored by the database manager.

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. To support commit sequences from different processes, the DB2 Connect concentrator must be enabled. To enable the DB2 Connect EE concentrator, set the database manager configuration parameter *max_logicagents* to a value greater then *maxagents*. Note that the DB2 Connect EE concentrator requires a DB2 Version 7.1 client to support XA commit sequences from different processes. For information about the SQL statements that are allowed in this environment, refer to the *Application Development Guide*. For information about the concentrator, refer to the *DB2 Connect User's Guide*.

If you will be updating host or AS/400 database servers, you will require DB2 Connect with the DB2 sync point manager (SPM) configured. Refer to one of the *Quick Beginnings* books for instructions.

Database Connection Considerations

The following topics are covered in this section:

- "RELEASE Statement"
- "Transactions Accessing Partitioned Databases"

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

In a partitioned database environment, user data may be distributed 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 transactions against a database in a partitioned database 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 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, described in "Understanding the Two-Phase Commit Process" on page 141. The principal 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 144) 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 indoubt transactions, there are actions you can take to manually resolve them. This manual process is sometimes referred to as ″making a heuristic decision″.

The LIST INDOUBT TRANSACTIONS command (using the WITH PROMPTING option), or the 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. To obtain indoubt transaction information from DB2 UDB on UNIX based systems, the Windows operating system, or OS/2, connect to the database and issue the LIST INDOUBT TRANSACTIONS WITH PROMPTING command, or the equivalent API. For information about this command or the related administrative APIs, refer to the *Command Reference* or the *Administrative API Reference*.

For indoubt transaction information with respect to host or AS/400 database servers, you have two choices:

- You can obtain indoubt information directly from the host or AS/400 server.

  To obtain indoubt information directly from DB2 for OS/390, invoke the DISPLAY THREAD TYPE(INDOUBT) command. Use the RECOVER command to make a heuristic decision. To obtain indoubt information directly from DB2 for OS/400, invoke the **wrkcmtdfn** command.
- You can obtain indoubt information from the DB2 Connect server used to access the host or AS/400 database server.

  To obtain indoubt information from the DB2 Connect server, first connect to the DB2 sync point manager by connecting to the DB2 instance represented by the value of the *spm_name* database manager configuration parameter. Then issue the LIST DRDA INDOUBT TRANSACTIONS WITH PROMPTING command to display indoubt transactions and to make heuristic decisions.

Use these commands (or related APIs) with *extreme caution*, and only as a last resort. The best strategy 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 against another participating database. Recovering from data integrity problems requires you to understand the application logic, to identify the data that was changed or rolled back, and then to perform a point-in-time recovery of the database, or manually undo or reapply the 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, heuristic operations are necessary. This situation could occur if the transaction manager will not be available for an extended period of time to perform 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 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.

Although there is no foolproof way to perform heuristic operations, the following provides some general guidelines:

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 the 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, 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 against the database; that is, using the LIST INDOUBT TRANSACTIONS command, either heuristically commit or heuristically roll back the transaction.
   - 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.
     – 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, indicated in output from the LIST INDOUBT TRANSACTIONS command. The heuristic forget function releases the log space occupied by an 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 for the transaction in this resource manager. In general a ″missing″ log record implies that the resource manager has rolled back the transaction.

Security Considerations

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 TXSeries CICS region is started, it is associated with the AIX user name under which it is defined. All the CICS Application Server processes are also being run under this TXSeries CICS ″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. 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 appear to the RM as a single program with many units of work from the same ″cics″ ID. Optionally, you can 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.

There is not much impact on static SQL statements, 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 ID, and not to the end user ID.

For dynamic statements, which have their access authentication done at run time, access privileges to the database objects must be granted to the server ID and not to the actual user of those objects. Instead of relying on the database to control the access of specific users, you must rely on the TP monitor system to determine which users can run which programs. The server ID must be granted all privileges that its SQL users require.

To determine who has accessed a database table or view, you can perform the following steps:

1. From the SYSCAT.PACKAGEDEP catalog view, obtain a list of all packages that depend on the table or view.
2. Determine the names of the server programs (for example, CICS programs) that correspond to these packages through the naming convention used in your installation.
3. Determine the client programs (for example, CICS transaction IDs) that could invoke these programs, and then use the TP monitor's log (for example, the CICS log) to determine who has run these transactions or programs, and when.

Configuration Considerations

You should consider the following configuration parameters when you are setting up your TP monitor environment:

- *tp_mon_name*

  This database manager configuration parameter identifies the name of the TP monitor product being used ("CICS", or "ENCINA", for example).

- *tpname*

  This database manager 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, using the APPC communications protocol. The value 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 database manager configuration parameter is not used for XA-coordinated transactions.

- *maxappls*

  This 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 applications that may be concurrently in the process of completing a two-phase commit or rollback. This sum should then be increased by the anticipated number of indoubt transactions that might exist at any one time. For more information about indoubt transactions, see "Recovering from Problems During Two-Phase Commit" on page 144.

  For a TP monitor environment (for example, TXSeries CICS), you may need to increase the value of the *maxappls* parameter. This would help to 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 value is `YES` (that is, enabled).

  A database containing indoubt transactions requires a restart database operation to start up. If *autorestart* is not enabled when the last connection to the database is dropped, the next connection will fail and require an explicit RESTART DATABASE invocation. This condition will exist until the indoubt transactions have been removed, either by the transaction manager's resync operation, or through a heuristic operation initiated by the administrator. When the RESTART DATABASE command is issued, a message is returned if there are any indoubt transactions in the database. The administrator can then use the LIST INDOUBT TRANSACTIONS command and other command line processor commands to find get 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*, with the following exceptions:

- Asynchronous services

  The XA specification allows the interface to use asynchronous services, so that the result of a request can be checked at a later time. The database manager requires that the requests be invoked in synchronous mode.
- Static registration

  The XA interface allows two ways to register an RM: static registration and dynamic registration. DB2 Universal Database supports only dynamic registration, which is more advanced and efficient. For more information about these two methods, see "Resource Managers (RM)" on page 151.
- Association Migration

  DB2 Universal Database does not support transaction migration between threads of control.

For information about xa_open and xa_close strings usage, see "xa_open and xa_close Strings Usage" on page 152.

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 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 Universal Database uses dynamic registration, and that the TM should not use association migration. Implicitly states that asynchronous operation is not supported.
version	Must be zero.

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 RM's **xa_** routines. An RM 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.

UNIX Based Systems and OS/2: DB2 UDB's switch can be obtained through either of the following two ways:

- Through one additional level of indirection. In a C program, this can be accomplished by defining the macro:

  ```
  #define db2xa_switch (*db2xa_switch)
  ```

  prior to using *db2xa_switch*.
- By calling **db2xacic**

  DB2 UDB provides this API, 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` (on UNIX based system) or `db2api.lib` (on OS/2).

Windows NT: 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 using *db2xa_switch*.
- If using the Microsoft Visual C++ compiler, *db2xa_switch* can be defined as:

  ```
  extern __declspec(dllimport) struct xa_switch_t db2xa_switch
  ```
- By calling **db2xacic**

DB2 UDB provides this API, 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, you must link your application with `db2api.lib`.

Example C Code: The following code illustrates the different ways in which the *db2xa_switch* can be accessed via a C program on any DB2 UDB platform. Be sure to link your application with the appropriate library.

```
   #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 ;
   }
```

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 effect. For more information about the First Failure Service Log, refer to the *Troubleshooting Guide*.

You should also consult the console message, TM error file, or other product-specific information about the external transaction processing software that you are using.

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. Following are some of the more common 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 user name or password is not authorized to connect to the database.
- Communications error.

Following is an example of an error log for an xa_open error (due to a missing xa_open string) generated on AIX:

```
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
```

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"
- "Configuring IBM TXSeries Encina"
- "Configuring BEA Tuxedo" on page 169
- "Configuring Microsoft Transaction Server" on page 171.

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 http://www.transarc.com/Library/documentation/websphere/WAS-EE/en_US/html/.

Host and AS/400 database servers can participate in CICS-coordinated transactions.

Configuring IBM TXSeries Encina

Following are the various APIs 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 through DB2 Connect. TXSeries documentation can be viewed online at http://www.transarc.com/Library/documentation/websphere/WAS-EE/en_US/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, a node directory entry must also be defined. You can perform the configuration using the 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 behavior is no special optimization. If *tp_mon_name* is set, the application must ensure that 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, ensure that the *tp_mon_name* value is `NONE` (or, through the CLP, that the value is set to `NULL`). The parameter can be updated through the Control Center or the CLP. The CLP command is:

   ```
   db2 update dbm cfg using tp_mon_name ENCINA
   ```

Configuring Encina for Each Resource Manager

To configure Encina for each resource manager (RM), 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 "db=inventdb,uid=user1,pwd=password1"
```

There is one resource manager configuration for each DB2 database, and each resource manager 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 user ID 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. For information about DB2's xa_open string, see "Setting Up a Database as a Resource Manager" on page 152.

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 UDB 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, you must use the DB2 Syncpoint Manager. Refer to the *DB2 Connect Enterprise Edition for OS/2 and Windows 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 lifetime of the transaction.
   - The application acts as a single-thread.

   Note: If you are using the ODBC or DB2 Call Level Interface, you must disable 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 Windows operating systems and OS/2, and `libdb2` on UNIX based systems).

   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 API. 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. Avoid using these transactions, if possible. If you cannot, ensure that SQL work is done in only one member of the Encina transaction family.

Configuring BEA Tuxedo

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 Application Development Client. 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. Set the PATH variable to include the DB2 program directories. Confirm that the Tuxedo server ID can connect to the DB2 databases.
3. Update the *tp_mon_name* database manager configuration parameter with the value `TUXEDO`.
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:db=sample,uid=db2_user,pwd=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. If the application is accessing data residing on DB2 for OS/390, DB2 for OS/400, or DB2 for VM&VSE, the DB2 Connect XA concentrator will be required. For configuration details and limitations, refer to the *DB2 Connect User's Guide*.
9. 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 later 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.

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.ibm.com/software/data/db2/library/, and search for a DB2 Universal Database ″Technote″ with the keyword ″MTS″.

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.

MTS Software Prerequisites

MTS support requires the DB2 Client Application Enabler (CAE) Version 5.2, or later, 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, you 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 (EE), either on your local machine or on a remote machine. DB2 Connect EE allows host or AS/400 database servers to participate in a multisite update transaction.

b. Ensure that your DB2 Connect EE server is enabled for multisite update. For information about enabling DB2 Connect for multisite updates, refer to 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 0)
- 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 applications 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)

Verifying the Installation

1. Configure your 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 through 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 multisite update using MTS-coordinated transactions:

- DB2 Universal Database Enterprise Edition Version 6 and later
- DB2 Universal Database Enterprise - Extended Edition Version 6 and later
- DB2 for OS/390
- DB2 for MVS
- DB2 for AS/400
- DB2 for VM & VSE

MTS Transaction Time-Out and DB2 Connection Behavior

You can set the transaction time-out value in the MTS Explorer tool. For more information, refer to the online *MTS Administrator Guide*.

If a transaction takes longer than the transaction time-out value (default value 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 request, the rollback request is 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 processing, the rollback request waits until the SQL statement finishes.

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 re-established 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 174).

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 the 32-bit Windows operating system fully supports connection pooling; therefore, this keyword is registered. Version 5.2 clients must install FixPack 3 (WR09024) or later.

The default value of 60 means that the connection will be pooled for 60 seconds before it 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 large amounts of system resource, including system memory and communications stack resources.

In addition, to ensure that the same connection is used between objects in the same transaction in a multiple processor machine, you must turn off ″multiple pool per processor″ support. To do this, copy the following registry setting into a file called `odbcpool.reg`, save it as a plain text file, and issue the command **odbcpool.reg**. The Windows operating system will import this registry setting.

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Connection Pooling]
"NumberOfPools"="1"
```

Without this keyword set to 1, MTS may pool connections in different pools, and hence will not reuse the same connection.

MTS Connection Pooling using ADO 2.1 and Later

If the MTS COM objects use ADO to access the database, you must turn off the OLEDB resource pooling so that the Microsoft OLEDB provider for ODBC (MSDASQL) will not interfere with ODBC connection pooling. This feature was initialized to `OFF` in ADO 2.0, but is initialized to `ON` in ADO 2.1. To turn OLEDB resource polling off, copy the following lines into a file called `oledb.reg`, save it as a plain text file, and issue the command **oledb.reg**. The Windows operating system will import these registry settings.

```
REGEDIT4

[HKEY_CLASSES_ROOT\CLSID\{c8b522cb-5cf3-11ce-ade5-00aa0044773d}]
@="MSDASQL"
"OLEDB_SERVICES"=dword:fffffffc
```

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 that there are two COM objects, COM1 and COM2, that connect to the same ODBC data source 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, 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 constraints.

To alleviate this problem, 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

Name	Default Value	Suggested Value
TcpMaxConnectRetransmission	3	6
TcpMaxDataRetransmission	5	8
TcpMaxRetransmissionAttempts	7	10
If the registry value is not defined, 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_id* and *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, select the **System DSN** tab, and then add the data source:
   a. Select IBM ODBC Driver, and then select **Finish**.
   b. When presented with the list of database aliases, choose the one that was specified previously.
   c. Select **OK**.
3. Use DB2 CLP to connect to a DB2 database under the ID *your_user_id*, as above.
   a. Bind the `db2cli.lst` file:

      ```
      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&VM). For example:

      ```
      db2 bind @C:\sqllib\bnd\@ddcsmvs.lst blocking all grant public
      ```

Otherwise, bind the db2ubind.lst file:

```
db2 bind @C:\sqllib\bnd\@db2ubind.lst blocking all grant public
```

c. 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.

Chapter 11. Introducing High Availability and Failover Support

Successful e-businesses depend on the uninterrupted availability of transaction processing systems, which in turn are driven by database management systems, such as DB2, that must be available 24 hours a day and 7 days a week ("24 x 7").

High Availability

High availability (HA) is the term that is used to describe systems that run and are available to customers more or less all the time. For this to occur:

- Transactions must be processed efficiently, without appreciable performance degradations (or even loss of availability) during peak operating periods. In a partitioned database environment, DB2 can take advantage of both intrapartition and interpartition parallelism to process transactions efficiently. *Intrapartition parallelism* can be used in an SMP environment to process the various components of a complex SQL statement simultaneously. *Interpartition parallelism* in a partitioned database environment, on the other hand, refers to the simultaneous processing of a query on all participating nodes; each node processes a subset of the rows in the table. For more information about parallelism, see "Types of Parallelism" on page 35.
- Systems must be able to recover quickly when hardware or software failures occur, or when disaster strikes. DB2 has an advanced continuous checkpointing system and a parallel recovery capability that allow for extremely fast crash recovery.

  The ability to recover quickly can also depend on having a proven backup and recovery strategy in place. For more information about recovery strategies, see the *Data Recovery and High Availability Guide and Reference*.
- Software that powers the enterprise databases must be continuously running and available for transaction processing. To keep the database manager running, you must ensure that another database manager can take over if it fails. This is called failover. *Failover* capability allows for the automatic transfer of workload from one system to another when there is hardware failure.

Failover protection can be achieved by keeping a copy of your database on another machine that is perpetually rolling the log files forward. *Log shipping* is the process of copying whole log files to a standby machine, either from an archive device, or through a user exit program running against the primary database. With this approach, the primary database is restored to the standby machine, using either the DB2 restore utility or the split mirror function. You can use the new suspended I/O support to quickly initialize the new database (see “High Availability through Online Split Mirror and Suspended I/O Support” on page 181). The secondary database on the standby machine continuously rolls the log files forward. If the primary database fails, any remaining log files are copied over to the standby machine. After a rollforward to the end of the logs and stop operation, all clients are reconnected to the secondary database on the standby machine.

Failover support can also be provided through platform-specific software that you can add to your system. For example:

- High Availability Cluster Multi-Processing, Enhanced Scalability, for AIX.

  For detailed information about HACMP/ES, see the *Data Recovery and High Availability Guide and Reference*, or the white paper entitled “IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES”, which is available from the “DB2 UDB and DB2 Connect Online Support” Web site (http://www.ibm.com/software/data/pubs/papers/).
- Microsoft Cluster Server, for Windows NT or Windows 2000.

  For detailed information about MSCS, see the *Data Recovery and High Availability Guide and Reference*.
- Sun Cluster, or VERITAS Cluster Server, for the Solaris Operating Environment.

  For information about Sun Cluster 2.x, see the *Data Recovery and High Availability Guide and Reference*; for information about Sun Cluster 3.0, see the white paper entitled “DB2 and High Availability on Sun Cluster 3.0”, which is available from the “DB2 UDB and DB2 Connect Online Support” Web site (http://www.ibm.com/software/data/pubs/papers/). For information about VERITAS Cluster Server, see the white paper entitled “DB2 and High Availability on VERITAS Cluster Server”, which is also available from the “DB2 UDB and DB2 Connect Online Support” Web site.
- Multi-Computer/ServiceGuard, for Hewlett-Packard.

  For detailed information about HP MC/ServiceGuard, see the white paper entitled “IBM DB2 EE v.7.1 Implementation and Certification With Hewlett-Packard’s MC/ServiceGuard High Availability Software”, which is available from the “DB2 UDB and DB2 Connect Online Support” Web site (http://www.ibm.com/software/data/pubs/papers/).

Failover strategies are usually based on clusters of systems. A *cluster* is a group of connected systems that work together as a single system. Each processor is known as a node within the cluster. Clustering allows servers to back each other up when failures occur, by picking up the workload of the failed server.

IP address takeover (or IP takeover) is the ability to transfer a server IP address from one machine to another when a server goes down; to a client application, the two machines appear at different times to be the same server.

Failover software may use *heartbeat monitoring* or *keepalive packets* between systems to confirm availability. Heartbeat monitoring involves system services that maintain constant communication between all the nodes in a cluster. If a heartbeat is not detected, failover to a backup system starts. End users are usually not aware that a system has failed.

The two most common failover strategies on the market are known as *idle standby* and *mutual takeover*, although the configurations associated with these terms may also be associated with different terms that depend on the vendor:

Idle Standby
: In this configuration, one system is used to run a DB2 instance, and the second system is "idle", or in standby mode, ready to take over the instance if there is an operating system or hardware failure involving the first system. Overall system performance is not impacted, because the standby system is idle until needed.

Mutual Takeover
: In this configuration, each system is the designated backup for another system. Overall system performance may be impacted, because the backup system must do extra work following a failover: it must do its own work plus the work that was being done by the failed system.

Failover strategies can be used to failover an instance, a partition, or multiple logical nodes.

High Availability through Online Split Mirror and Suspended I/O Support

Suspended I/O supports continuous system availability by providing a full implementation for online split mirror handling; that is, splitting a mirror without shutting down the database. A *split mirror* is an "instantaneous" copy of the database that can be made by mirroring the disks containing the data, and splitting the mirror when a copy is required. *Disk mirroring* is the process of writing all of your data to two separate hard disks; one is the mirror of the other. *Splitting* a mirror is the process of making a backup copy of the mirror.

If you would rather not back up a large database using the DB2 backup utility, you can make copies from a mirrored image by using suspended I/O and the split mirror function. This approach also:

- Eliminates backup operation overhead from the production machine
- Represents a fast way to clone systems

- Represents a fast implementation of idle standby failover. There is no initial restore operation, and if a rollforward operation proves to be too slow, or encounters errors, reinitialization is very fast.

The **db2inidb** command initializes the split mirror so that it can be used:

- For making a clone database

  A read-only clone of the primary database can be used, for example, to create reports.
- As a standby database
- As a backup image

This command can only be issued against the split-off mirror, and the split-off mirror must first run **db2inidb** before it can be used (see the *Data Recovery and High Availability Guide and Reference*).

In a partitioned database environment, the **db2inidb** command must be run on every partition before the split image from any of the partitions can be used. The tool can be run on all partitions simultaneously.

Making a Clone Database

A database clone can represent an offline "backup" of the primary (live) database. You cannot, however, back up the cloned database, restore this image on the original system, and roll forward through log files produced on the original system.

To clone a database, follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. Attach to the mirrored database from another machine.
5. Start the database instance:
   ```
   db2start
   ```
6. Initialize the mirrored database as a clone of the primary database:
   ```
   db2inidb database_alias as snapshot
   ```

   Note: This command will roll back transactions that are in flight when the split occurs.

Using the Split Mirror as a Standby Database

As the mirrored (standby) database continually rolls forward through the logs, new logs that are being created by the primary database are continually fetched from the primary system. To use the split mirror as a standby database, follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. Attach the mirrored database to another instance.
5. Put the mirrored database in rollforward pending state:
   ```
   db2inidb database_alias as standby
   ```

   If you have only DMS table spaces (database managed space), you can take a full database backup to offload the overhead of taking a backup on the production database.
6. Set up a user exit program to retrieve the log files from the primary system.
7. Roll the database forward to the end of the logs.
8. Continue retrieving log files, and rolling the database forward to the end of the logs until the primary database goes down.

Using the Split Mirror as a Backup Image

To use the split mirror as a "backup image", follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. A failure occurs on the primary system, necessitating a restore from backup.
5. Use operating system-level commands to copy the split-off data over the primary system. Do not copy the split-off logs, because the primary logs will be needed for rollforward recovery.
6. Start the primary database instance:
   ```
   db2start
   ```
7. Initialize the primary database:
   ```
   db2inidb database_alias as mirror
   ```

8. Roll the primary database forward to the end of the logs.

Part 4. Appendixes

Appendix A. Naming Rules

Go to the section that describes the naming rules that you require information on:

- "General Naming Rules"
- "Object Naming Rules"
- "How Case-Sensitive Values Are Preserved in a Federated System" on page 191

General Naming Rules

Unless otherwise specified, all names can include the following characters:

- A through Z. When used in most names, characters A through Z are converted from lowercase to uppercase.
- 0 through 9
- @, #, $, and _ (underscore)

Names cannot begin with a number or with the underscore character.

Do not use SQL reserved words to name tables, views, columns, indexes, or authorization IDs. For a list of SQL reserved words, see the *SQL Reference*.

There are other special characters that might work separately depending on your operating system and where you are working with DB2. However, while they might work, there is no guarantee that they will work. It is not recommended that you use these other special characters when naming objects in your database.

Object Naming Rules

All objects follow the General Naming Rules. In addition, some objects have additional restrictions shown below.

Table 24. Database, Database Alias and Instance Naming Rules

Objects	**Guidelines**
• Databases • Database aliases • Instances	• Database names must be unique within the location in which they are cataloged. On UNIX-based implementations of DB2, this location is a directory path, while on Windows implementations, it is a drive letter. • Database alias names must be unique within the system database directory. 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. • Database, database alias and instance names can have up to 8 bytes. • On Windows NT and Windows 2000 systems, no instance can have the same name as a service name. **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 language.

Table 25. Database Object Naming Rules

Objects	**Guidelines**
• Aliases • Buffer pools • Columns • Event monitors • Indexes • Methods • Nodegroups • Schemas • Stored procedures • Tables • Table spaces • Triggers • UDFs • UDTs • Views	Can contain up to 18 bytes *except* for the following: • Table names (including view names, summary table names, alias names, and correlation names), which can contain up to 128 bytes • Column names, which can contain up to 30 bytes • Schema names, which can contain up to 30 bytes • Object names can also include: – valid accented characters (such as ö) – multibyte characters, except multibyte spaces (for multibyte environments)

Additional Information about Schema Names

- Tables with schema names longer than 18 bytes cannot be replicated.
- User-defined types (UDTs) cannot have schema names longer than 8 bytes.

- The following schema names are reserved words and must not be used: SYSCAT, SYSFUN, SYSIBM, SYSSTAT.
- To avoid potential migration problems in the future, do not use schema names that begin with SYS. The database manager will not allow you to create triggers, user-defined types or user-defined functions using a schema name beginning with SYS.
- It is recommended that you not use SESSION as a schema name. Declared temporary tables must be qualified by SESSION. It is therefore possible to have an application declare a temporary table with a name identical to that of a persistent table, in which case the application logic can become overly complicated. Avoid the use of the schema SESSION, except when dealing with declared temporary tables.

Table 26. User, User ID and Group Naming Rules

Objects	Guidelines
• Group names • User names • User IDs	• Group names can contain up to 8 bytes. • User IDs on UNIX-based systems can contain up to 8 characters. • User names on Windows can contain up to 30 characters. Windows NT and Windows 2000 currently have a practical limit of 20 characters. • When using DCE authentication, user names have a limit of 8 characters. • When not using DCE or Client authentication, non-Windows 32-bit clients connecting to Windows NT and Windows 2000 with user names longer than 8 characters are supported when the user name and password are specified explicitly. • Names and IDs cannot: – Be USERS, ADMINS, GUESTS, PUBLIC, LOCAL or any SQL reserved word listed in the *SQL Reference*. – Begin with IBM, SQL or SYS. – Include accented characters. On UNIX-based systems, groups and users can have the same name. For the GRANT statement, you must specify whether you are referring to a group or to 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 Windows NT, local groups, global groups, and users cannot have the same name. On OS/2, groups and users cannot have the same name. **Notes:** 1. Some operating systems allow case sensitive user IDs and passwords. You should check your operating system documentation to see if this is the case. 2. The authorization ID returned from a successful CONNECT or ATTACH is truncated to 8 characters. An ellipsis (...) is appended to the authorization ID and the SQLWARN fields contain warnings to indicate truncation. For more information, see the CONNECT statement in the *SQL Reference*.

Additional Information about Passwords

You may be required to perform password maintenance tasks. Since such tasks are required at the server, and many users are not able or comfortable working with the server environment, performing 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, use the CONNECT statement to change the password as follows:

```
CONNECT TO <database> USER <userid> USING <password>
   NEW <new_password> CONFIRM <new_password>
```

The “Password change” dialog of the DB2 Client Configuration Assistant (CCA) can also be used to change the password. For more information about these methods of changing the password, refer to the *SQL Reference* and the CCA online help.

Table 27. Federated Database Object Naming Rules

Objects	Guidelines
• Function mappings • Index specifications • Nicknames • Servers • Type mappings • User mappings • Wrappers	• Nicknames, mappings, index specifications, servers, and wrapper names cannot exceed 128 bytes. • Server and nickname options and option settings are limited to 255 bytes. • Names for federated database objects can also include: – Valid accented letters (such as ö) – Multibyte characters, except multibyte spaces (for multibyte environments)

Using Delimited Identifiers in Object Names

Keywords can be used. If a keyword is used in a context where it could also be interpreted as an SQL keyword, it must be specified as a delimited identifier.

Using delimited identifiers, it is possible to create an object that violates these naming rules; however, subsequent use of the object could result in errors. For example, if you create 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. For information about delimited identifiers, see the ″SQL Identifiers″ section of the *SQL Reference*.

How Case-Sensitive Values Are Preserved in a Federated System

With distributed requests, you sometimes need to specify identifiers and passwords that are case sensitive at the data source. To ensure that the case is correct when they are passed to the data source, follow these guidelines:

- Specify identifiers and passwords in the required case, and enclose them in double quotation marks.
- If you are specifying a user ID, set the *fold_id* server option to ″n″ (“No, don’t change case”) for the data source. If you are 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 uppercase”). 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″ in the *Administration Guide: Implementation* .

- If you enclose a case sensitive identifier or a password in double quotation marks at an operating system command prompt, you must ensure that the system parses the double quotation marks correctly. To do this:
  - On a UNIX-based operating system, enclose the statement in single quotation marks.
  - On the Windows NT operating system, precede each quotation mark with a backslash.

For example, many delimited identifiers in DB2 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`.

At the command prompt for a UNIX-based system, you would type:

```
db2 'create nickname nick1 for norbase."my_schema"."wkly_sal"'
```

At a Windows NT command prompt, you would type:

```
db2 create nickname nick1 for norbase.\"my_schema\".\"wkly_sal\"
```

If you enter the statement from a DB2 command prompt (interactive mode), or if you specify it in an application program, you do not need the single quotation marks or the backslashes. For example, at the DB2 command prompt on either a UNIX-based system or Windows NT, you would type:

```
create nickname nick1 for norbase."my_schema"."wkly_sal"
```

Appendix B. Planning Database Migration

This section provides you with an overview of the migration process. Note that DB2 UDB Version 6 databases do not need to be migrated to Version 7. Detailed information about migrating your DB2 UDB Version 5.*x* databases can be found in the *Quick Beginnings* manual for your operating system.

When you migrate your database:

- The following database entities are migrated:
  - Database configuration file
  - Database system catalog tables
  - Database directories
  - Database log file header
- 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 the SYSFUN.DIFFERENCE scalar function is dropped and recreated during database migration.
- 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.

Migration Considerations

To successfully migrate a database created with a previous version of the database manager, you must consider the following:

- "Migration Restrictions" on page 194
- "Security and Authorization" on page 194
- "Storage Requirements" on page 194
- "Release-to-Release Incompatibilities" on page 194

Migration Restrictions

There are certain pre-conditions or restrictions that you should be aware of before attempting to migrate your database to Version 7:

- Migration is only supported from V5.x or V6. Migration from DB2 V1.2 Parallel Edition is not supported. Earlier versions of DB2 (Database Manager) must be migrated to V5.x or V6 before being migrated to V7.
- Issuing the migration command from a V7 client to migrate a database to a V7 server is supported; however, issuing the migration command from an older DB2 client to migrate a database to a V7 server is not supported.
- Migration between platforms is not supported.
- User objects within your database cannot have V7 reserved schema names as object qualifiers. These reserved schema names include: SYSCAT, SYSSTAT, and SYSFUN.
- User-defined distinct types using the names BIGINT, REAL, DATALINK, or REFERENCE must be renamed before migrating the database.
- 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
  - Transaction inconsistent
- Restoration of down-level (V5.x or V6) database backups is supported, but the 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 the new catalogs during the migration. 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 that the database catalog currently occupies. If there is not enough disk space, migration fails.

If your SYSCAT table space is an SMS type of table space, you should also consider updating the database configuration parameters that are associated with the log files. You should increase the values of *logfilsiz*, *logprimary*, and *logsecond* to prevent the space for these log files from running out (SQL1704N with reason code 3). If this happens, increase the log space parameters, and re-issue the MIGRATE DATABASE command.

Release-to-Release Incompatibilities

Consider the impact of incompatibilities between the two versions of the product when planning to migrate a database.

To take advantage of Version 7 enhancements, you should tune your database and database manager configuration after migrating your databases. To facilitate this, you can record and compare configuration parameter values from before and after migration. (For a description of the GET DATABASE CONFIGURATION command and the GET DATABASE MANAGER CONFIGURATION command, refer to the *Command Reference*.)

Migrating a Database

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. Verify that there are no unresolved issues that pertain to "Migration Restrictions" on page 194.
2. Disconnect all applications and end users from each database being migrated (use the LIST APPLICATIONS command, or the FORCE APPLICATIONS command, as necessary).
3. Use the DB2CKMIG pre-migration utility to determine if the database can be migrated (for detailed information about using this utility, see the *Quick Beginnings* book for your platform). Note that on Windows NT or OS/2, you are prompted to run this tool during installation, but on UNIX based systems, this tool is invoked automatically during instance migration.
4. Back up your database.

   Migration is not a recoverable process. If you back up your database before the Version 6 reserved schema names are changed, you will not be able to restore the database using DB2 UDB Version 7. To restore the database, you will have to use your previous version of the database manager.

   Attention! If you do not have a backup of your database, and the migration fails, you will have no way of restoring your database using DB2 UDB Version 7, or your previous version of the database manager.

   You should also be aware that any database transactions done between the time the backup was taken and the time that the upgrade to Version 7 is completed are not recoverable. That is, if at some time following the completion of the installation and migration to Version 7, the database needs to be restored (to a Version 7 level), the logs written before Version 7 installation cannot be used in roll-forward recovery.

MIGRATION:

5. Migrate the database using one of the following:

- The MIGRATE DATABASE command
- The RESTORE DATABASE command, when restoring a full backup of the database
- The sqlemgdb - Migrate Database API.

On OS/2: The DB2CIDMG migration utility, which works in a Configuration/Installation/Distribution (CID) architecture environment, is only available on DB2 for OS/2. It permits remote unattended installation and configuration on LAN-based workstations. You must have NetView DM/2 on your LAN to use CID migration.

On UNIX based systems: The *Quick Beginnings* book for your platform describes what to do if you do not want to migrate all databases in a given instance.

POST-MIGRATION:

6. Optionally, use the DB2UIDDL utility to facilitate the management of a staged migration of unique indexes on your own schedule. (DB2 Version 5 databases that were created in Version 5 do not require this tool to take advantage of deferred uniqueness checking, because all unique indexes created in Version 5 have these semantics already. However, for databases that were previously migrated to Version 5, these semantics are not automatic, unless you use the DB2UIDDL utility to change the unique indexes.) This utility generates CREATE UNIQUE INDEX statements for unique indexes on user tables, and writes them to a file. Running this file as a DB2 CLP command file results in the unique index being converted to Version 7 semantics. For detailed information about using this utility, refer to one of the *Quick Beginnings* books.
7. Optionally, issue the RUNSTATS command against tables that are particularly critical to the performance of SQL queries. Old statistics are retained in the migrated database, and are not updated unless you invoke the RUNSTATS command.
8. Optionally, use the DB2RBIND utility to revalidate all packages, or allow package revalidation to occur implicitly when a package is first used.
9. Optionally, migrate Explain tables if you are planning to use them in Version 7. For more information, see "SQL Explain Facility" in the *Administration Guide: Performance*.
10. Tune your database and database manager configuration parameters to take advantage of Version 7 enhancements.

Appendix C. Incompatibilities Between Releases

This section identifies the incompatibilities that exist between DB2 Universal Database and previous releases of DB2.

An *incompatibility* is a part of DB2 Universal Database that works differently than it did in a previous release of DB2. If used in an existing application, it will produce an unexpected result, necessitate a change to the application, or reduce performance. In this context, ″application″ refers to:

- Application program code
- Third-party utilities
- Interactive SQL queries
- Command or API invocation.

Incompatibilities introduced with DB2 Universal Database Version 6 and Version 7 are described. They are grouped according to the following categories:

- System Catalog Views
- Application Programming
- SQL
- Database Security and Tuning
- Utilities and Tools
- Connectivity and Coexistence
- Configuration Parameters.

Each incompatibility section includes a description of the incompatibility, the symptom or effect of the incompatibility, and possible resolutions. There is also an indicator at the beginning of each incompatibility description that identifies the operating system to which the incompatibility applies:

WIN	Microsoft Windows platforms supported by DB2
UNIX	UNIX based platforms supported by DB2
OS/2	OS/2

Note: As of DB2 Universal Database Version 6, Version 1.x and Version 2.x clients, including the clients packaged with DB2 Parallel Edition Version 1.2 servers, are no longer supported.

DB2 Universal Database Planned Incompatibilities

This section describes future incompatibilities that users of DB2 Universal Database should keep in mind when coding new applications, or when modifying existing applications. This will facilitate migration to future versions of DB2 UDB.

Read-only Views in a Future Version of DB2 Universal Database

WIN	UNIX	OS/2

Change
The system catalog views will be read-only views. The SYSSTAT views will continue to be updatable.

Symptom
UPDATE statements that 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

WIN	UNIX	OS/2

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

WIN	UNIX	OS/2

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 7 Incompatibilities

This section identifies incompatibilities introduced in DB2 Universal Database Version 7.

Application Programming

Query Patroller Universal Client

WIN	UNIX	OS/2

Change: This new version of the client application enabler (CAE) will only work with Query Patroller Server Version 7, because there are new stored procedures. CAE is the application interface to DB2 through which all applications must eventually pass to access the database.

Symptom: If this CAE is run against a back-level server, message SQL29001 is returned.

Object Transform Functions and Structured Types

WIN	UNIX	OS/2

Change: There is a minor and remotely possible incompatibility between a pre-Version 7 client and a Version 7 server that relates to changes that have been made to the SQLDA. As described in the *Application Development Guide*, byte 8 of the second SQLVAR can now take on the value X'12' (in addition to the values X'00' and X'01'). Applications that do not anticipate the new value may be affected by this extension.

Resolution: Because there may be other extensions to this field in future releases, developers are advised to only test for explicitly defined values.

Versions of Class and Jar Files Used by the JVM

WIN	UNIX	OS/2

Change: Previously, once a Java stored procedure or user-defined function (UDF) was started, the Java Virtual Machine (JVM) locked all files given in the CLASSPATH (including those in `sqllib/function`). The JVM used these files until the database manager was stopped. Depending on the environment in which you run a stored procedure or UDF (that is, depending on the value of the *keepdari* database manager configuration parameter, and whether or not the stored procedure is fenced), refreshing classes will let you replace class and jar files without stopping the database manager. This is different from the previous behavior.

Changed Functionality of Install, Replace, and Remove Jar Commands

WIN	UNIX	OS/2

Change: Previously, installation of a jar caused the flushing of all DARI (Database Application Remote Interface) processes. This way, a new stored procedure class was guaranteed to be picked up on the next call. Currently, no jar commands flush DARI processes. To ensure that classes from newly installed or replaced jars are picked up, you must explicitly issue the SQLEJ.REFRESH_CLASSES command.

Another incompatibility introduced by not flushing DARI processes is the fact that for fenced stored procedures, with the value of the *keepdari* database manager configuration parameter set to "YES", clients may get different versions of the jar files. Consider the following scenario:

1. User A replaces a jar and does not refresh classes.
2. User A then calls a stored procedure from the jar. Assuming that this call uses the same DARI process, User A will get an old version of the jar file.
3. User B calls the same stored procedure. This call uses a new DARI, which means that the newly created class loader will pick up the new version of the jar file.

In other words, if classes are not refreshed after jar operations, a stored procedure from different versions of jars may be called, depending on which

DARI processes are used. This differs from the previous behavior, which ensured (by flushing DARI processes) that new classes were always used.

32-bit Application Incompatibility

	UNIX	

Change: 32-bit executables (DB2 applications) will not run against the new 64-bit database engine.

Symptom: The application fails to link. When you attempt to link 32-bit objects against the 64-bit DB2 application library, an operating system linker error message is displayed.

Resolution: The application must be recompiled as a 64-bit executable, and relinked against the new 64-bit DB2 libraries.

Changing the Length Field of the Scratchpad

WIN	UNIX	OS/2

Change: Any user-defined function (UDF) that changes the length field of the scratchpad passed to the UDF will now receive SQLCODE -450.

Symptom: A UDF that changes the length field of the scratchpad fails. The invoking statement receives SQLCODE -450, with the schema and the specific name of the function filled in.

Resolution: Rewrite the UDF body to not change the length field of the scratchpad.

SQL

Applications that Use Regular Tables Qualified by the Schema SESSION

WIN	UNIX	OS/2

Change: The schema SESSION is the only schema allowed for temporary tables, and is now used by DB2 to indicate that a SESSION-qualified table may refer to a temporary table. However, SESSION is not a keyword reserved for temporary tables, and can be used as a schema for regular base tables. An application, therefore, may find a SESSION.T1 real table and a SESSION.T1 declared temporary table existing simultaneously. If, when a package is being bound, a static statement that includes a table reference qualified (explicitly or implicitly) by ″SESSION″ is encountered, neither a section nor dependencies for this statement are stored in the catalogs. Instead, this section will need to be incrementally bound at run time. This will place a copy of the section in the dynamic SQL cache, where the cached copy will be private only to the unique instance of the application. If, at run time, a declared temporary table matching the table name exists, the declared temporary table is used, even if a permanent base table of the same name exists.

Symptom: In Version 6 (and earlier), any package with static statements involving tables qualified by SESSION would always refer to a permanent base table. When binding the package, a section, as well as relevant dependency records for that statement, would be saved in the catalogs. In Version 7, these statements are not bound at bind time, and could resolve to a declared temporary table of the same name at run time. Thus, the following situations can arise:

- Migrating from Version 5. If such a package existed in Version 5, it will be bound again in Version 6, and the static statements will now be incrementally bound. This could affect performance, because these incrementally bound sections behave like cached dynamic SQL, except that the cached dynamic section cannot be shared among other applications (even different instances of the same application executable).
- Migrating from Version 6 to Version 7. If such a package existed in Version 6, it will not necessarily be bound again in Version 7. Instead, the statements will still execute as regular static SQL, using the section that was saved in the catalog at original bind time. However, if this package is rebound (either implicitly or explicitly), the statements in the package with SESSION-qualified table references will no longer be stored, and will require incremental binding. This could degrade performance.

To summarize, any packages bound in Version 7 with static statements referring to SESSION-qualified tables will no longer perform like static SQL, because they require incremental binding. If, in fact, the application process issues a DECLARE GLOBAL TEMPORARY TABLE statement for a table that has the same name as an existing SESSION-qualified table, view, or alias, references to those objects will always be taken to refer to the declared temporary table.

Resolution: If possible, change the schema names of permanent tables so that they are not ″SESSION″. Otherwise, there is no recourse but to be aware of the performance implications, and the possible conflict with declared temporary tables that may occur.

The following query can be used to identify tables, views, and aliases that may be affected if an application uses temporary tables:

```
select tabschema, tabname from SYSCAT.TABLES where tabschema = 'SESSION'
```

The following query can be used to identify Version 7 bound packages that have static sections stored in the catalogs, and whose behavior might change if the package is rebound (only relevant when moving from Version 6 to Version 7):

```
select pkgschema, pkgname, bschema, bname from syscat.packagedep
   where bschema = 'SESSION' and btype in ('T', 'V', 'I')
```

Utilities and Tools

Data Links File Manager and File System Filter on Solaris

	UNIX	

Change: Data Links File Manager and File System Filter are not supported on Solaris OS 2.5.1.

db2set on AIX and Solaris

	UNIX	

Change: The command ″db2set -ul (user level)″ and its related functions are not ported to AIX or Solaris.

Data Links File System and Norton Utilities**

WIN		

Change: The Windows NT Data Links File System is incompatible with Norton Utilities.

Symptom: When a file is deleted from a drive controlled by DLFS, a kernel exception results: error 0x1E (Kernel Mode Exception Not Handled). The exception being 0xC00000005 (Access Violation).

Explanation: This access violation happens because the Norton Utilities driver is loaded after the DLFS filter driver is loaded.

Resolution: You can work around this by loading the DLFSD driver after the Norton Utilities driver is loaded. Change the DLFSD driver startup to manual by clicking on Start; then select Settings—> Control Panel—> Devices—> DLFSD and set it to manual.

You can create a batch file that loads the DLFSD driver and the DLFM Service on system startup. The contents of the batch file are as follows:

```
net start dlfsd
net start "dlfm service"
```

| Name this batch file start_dlfs.bat, and copy it into the
| **WINNT\Profiles\Administrator\Start Menu\Programs\Startup** directory.
| Only an administrator has the privilege to load the DLFS filter driver and the
| DLFM service.

Connectivity and Coexistence

32-bit Client Incompatibility

WIN	UNIX	OS/2

Change: 32-bit clients cannot attach to instances or connect to databases on 64-bit servers.

Symptom: If both the client and the server are running Version 7 code, SQL1434N is returned; otherwise, the attachment or connection fails with SQLCODE -30081.

Resolution: Use 64-bit clients.

DB2 Universal Database Version 6 Incompatibilities

This section identifies incompatibilities introduced in DB2 Universal Database Version 6.

System Catalog Views

System Catalog Views in DB2 Universal Database Version 6

WIN	UNIX	OS/2

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 type code ″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 type code. In Version 5.2, the type code ″T″ was used for all tables, and ″V″ was used for all views. In Version 6, untyped tables will continue to have a type code of ″T″ and typed tables will have a new type code of ″U″. Similarly, untyped views will continue to have a type code of ″V″ and typed views will have a new type code 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 type code 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 type codes ″T″, ″U″, ″V″, and ″W″ should be used. If the tool or application needs a physical view of tables, including hierarchy tables, then type codes ″T″ and ″H″ should be used.

Primary and Foreign Key Column Names in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: Data type change to two SYSCAT.REFERENCES columns, PK_COLNAMES and FK_COLNAMES, from VARCHAR(320) to VARCHAR(640).

Symptom: Primary key 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 a 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 names following the column whose length (*n*) is greater than 18 will be shifted *n*-18 bytes to the right. As well, if the list of column names exceeds 640 bytes, the column will contain the empty string.

Resolution: The SYSCAT.KEYCOLUSE view 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

WIN	UNIX	OS/2

Change: View text in the SYSCAT.VIEWS column TEXT will no longer 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 the application.

Explanation: Tools or applications that were coded to expect no more than 3600 (or perhaps 3900) bytes returned from the TEXT column at one time are not handling the increased size of this field. The mechanism for retrieving multiple rows and reconstructing the view text using the SEQNO field is no longer necessary. The SEQNO value will always be 1.

Resolution: Change the tool or application to be able to handle values from the TEXT column that 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

WIN	UNIX	OS/2

Change: Statement text in the SYSCAT.STATEMENTS column TEXT will no longer 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 the application.

Explanation: Tools or applications that were coded to expect no more than 3600 (or perhaps 3900) bytes returned from the TEXT column at one time are not handling the increased size of this field. The mechanism for retrieving multiple rows and reconstructing the statement text using the SEQNO field is no longer necessary. The SEQNO value will always be 1.

Resolution: Change the tool or application to be able to handle values from the TEXT column that 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

WIN	UNIX	OS/2

Change: The SYSCAT.INDEXES column COLNAMES data type is changed from VARCHAR(320) to VARCHAR(640).

Symptom: Column names are missing from an index.

Explanation: Tools or applications coded to retrieve data from a column with data type VARCHAR(320) cannot handle the increased size of this field.

Resolution: The SYSCAT.INDEXCOLUSE view contains the list of columns that make up an index, and should be used instead of the COLNAMES column. Alternatively, remove a column from the index, or reduce the size of the column name, so that the list of column names (with the leading + or −) will fit within 320 bytes.

SYSCAT.CHECKS Column TEXT in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: CHECKS Column TEXT data type is changed from CLOB(32K) to CLOB(64K).

Symptom: Check constraint clause is incomplete.

Explanation: Tools or applications coded to retrieve data from a column with data type CLOB(32K) cannot handle the increased size of this field.

Resolution: Change the tool or application to be able to handle values from the TEXT column that are longer than 32 KB. Alternatively, rewrite the check constraint clause to use fewer characters, so that it will fit within 32 KB.

Column Data Type to BIGINT in DB2 Universal Database Version 6

WIN	UNIX	OS/2

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 coded to retrieve data from a column with data type INTEGER cannot handle the increased size of this field.

Resolution: Change the tool or application to be able to handle values that are greater than the maximum, or less than the minimum value that can be stored in an INTEGER field. Alternatively, change the underlying structure or SQL code that causes the value to fall outside of what can be represented in an INTEGER field.

Column Mismatch in DB2 Universal Database Version 6

WIN	UNIX	OS/2

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 last.

Resolution: Explicitly name the columns in the select list instead of coding ″SELECT *″.

SYSCAT.COLUMNS and SYSCAT.ATTRIBUTES in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: SYSCAT.COLUMNS and SYSCAT.ATTRIBUTES now contain entries for inherited columns and attributes.

Symptom: Queries against SYSCAT.COLUMNS to retrieve the columns of a typed table or view, and queries against SYSCAT.ATTRIBUTES to retrieve the attributes of a structured type, may return more rows in Version 6 than in Version 5.2 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 Version 6, 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

WIN	UNIX	OS/2

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 running the CLP script called `objcatdp.db2`, found under the `misc` subdirectory of the `sqllib` directory.

You can also create your own set of OBJCAT views that are equivalent to the catalog views supported in Version 5.2.

In version 5.2, ″Appendix E″ of the *SQL Reference* 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

WIN	UNIX	OS/2

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 by code ″H″. In Version 6, these dependencies are denoted by code ″O″.

Resolution: Revise these queries to search for code ″O″.

SYSIBM Base Catalog Tables in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: Following are changes to the SYSIBM base catalog tables, which you may still be using instead of the SYSCAT views:

- Deleted fields (but still in the SYSCAT views):
  - SYSSTMT.SEQNO
  - SYSVIEWS.SEQNO
- Renamed catalog table: SYSTRIGDEP changed to SYSDEPENDENCIES. As well, the columns BCREATOR and DCREATOR were renamed to BSCHEMA and DSCHEMA, respectively. The view SYSCAT.TRIGDEP did not change.

- 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
- 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

WIN	UNIX	OS/2

Change: Maximum possible size of VARCHAR (VARGRAPHIC) data type has increased from 4000 characters (2000 double byte characters) to 32672 characters (16336 double byte characters) in Version 6.

Symptom: An application that uses fixed length buffers of 4000 bytes for a VARCHAR (VARGRAPHIC) data type has the potential for buffer overwrite or truncation, if it fetches a VARCHAR field that is longer than 4000 bytes into a buffer that is too small. The CLI function - `SQLGetTypeInfo()` now returns the size of VARCHAR as 32672. CLI applications that use this value in table DDLs may get errors because table spaces of sufficient page size are not available. For more information about table space page size, see “User Table Data” on page 93.

Resolution: When coding the application, it is recommended that you first describe the columns of the result set (using the DESCRIBE statement), and then use buffers whose size is based on the length returned from the DESCRIBE statement.

Java Programming Positioned UPDATE and DELETE in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: When programming 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, in which 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 in the package. Grant the correct privileges and then rebind the package.

Syntax Change in FOR UPDATE Clause in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: In Version 5.2, the FOR UPDATE clause in a SELECT statement can be used in an SQLJ program to identify the columns that can be updated in subsequent positioned UPDATE statements. The syntax has 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.

For more information about positioned iterator declarations, refer to the *Application Development Guide*.

Character Name Sizes in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: DB2 Universal Database Version 6 supports 128-byte table, view, and alias names, and 30-byte column names. Previous 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 was CHAR(8), and is now VARCHAR(128).

Symptom: If applications that were developed before Version 6 are run against a Version 6 database that does not use the longer limits, application behavior should not change at all. However, running these applications against a Version 6 database that *does* use longer names could result in certain side effects, depending on how these applications were coded.

Following 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. It will assume (incorrectly) that truncation will never 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 on the basis of 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 or column names may have increased. If other application logic operates on the assumption that column names are limited to 18 bytes, longer names that are returned may be handled in an unexpected way; for example, the display of longer column names may be truncated at 18 bytes.
- Since the SQLCA token field (`sqlerrmc`) is limited to 70 bytes, existing applications that attempt to insert a row into a table may be affected. In response to error SQL0204N, such applications determine the name of the table from the SQLCA `sqlerrmc` field, and then perform some operations based on that object name. With earlier versions of DB2, the table or schema

identifier limit guaranteed that the entire table name would be included in the SQLCA. This is not the case in Version 6.

- An application using a back-level API 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 when 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 recode the application to handle longer table and column names. Otherwise, ensure that these applications are not run against Version 6 databases that use greater than 18-byte names.

PC/IXF Format Changes in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: DB2 Universal Database Version 6 supports 128-byte table, view, and alias names, and 30-byte column names. Previous support was for 18-byte names for each of these entities.

Symptom: A DB2 Universal Database Version 5 client cannot import a PC/IXF file that was exported by a DB2 Universal Database Version 6 client (error SQL3059N). A PC/IXF file (exported from a DB2 Universal Database Version 6 client) cannot be loaded into a DB2 Universal Database Version 5 database (error SQL3059N).

Resolution: Use compatible versions of DB2 Universal Database when importing or loading PC/IXF data.

SQLNAME in a Non-doubled SQLVAR in DB2 Universal Database Version 6

WIN	UNIX	OS/2

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; for system-generated names and for user-specified column names specified in an ″AS″ clause, ″0x00″ is also placed in the 30th byte.

In Version 6, ″0xFF″ is returned in the 30th byte only if the name is system-generated.

Symptom: Any applications that rely on the 30th byte of the SQLNAME field 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 that field 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 migrating to a new version of DB2 UDB, 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 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. For information about these keywords, refer to the *CLI Guide and Reference*.

Event Monitor Output Stream Format in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: Event monitor output streams have no version control. As a result, adding support for greater than 18-byte table names requires moving to an output stream format.

Symptom: Applications that parse the event monitor output streams will no longer work properly.

Resolution: There are two options:
- Update the application to use the new data stream.

- Set the registry variable

  ```
  DB2OLDEVMON=evmonname1,evmonname2,...
  ```

  where *evmonname* is the name of the event monitor that you want written in the old data format. Note that any new fields in the event monitor will not be accessible under the old data format.

SQL

DATALINK Columns in DB2 Universal Database Version 6

	UNIX	

Change: DATALINK values inserted under 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

WIN	UNIX	OS/2

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 in Version 6. This occurs because the built-in functions (under the SYSIBM schema) are usually resolved before functions in the SYSFUN schema are resolved.

Resolution: No action is required. The built-in function is usually preferred over the function in the SYSFUN schema. The previous version behavior can be restored by switching the SQL path (so that SYSFUN precedes SYSIBM), but performance will be degraded. The previous version function can also 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 recreating the view, summary table, trigger, or constraint.

SET INTEGRITY replaces SET CONSTRAINTS

WIN	UNIX	OS/2

Change: The SET CONSTRAINTS statement has been replaced by the SET INTEGRITY statement. For backwards compatibility, both statements are accepted in DB2 Version 6 and Version 7.

SYSTABLE Column Change With New Integrity State in DB2 Universal Database Version 6

WIN	UNIX	OS/2

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 the CONST_CHECKED column changed to an ″N″ state when a SET INTEGRITY ... OFF statement was run. The ″U″ state now changes to a ″W″ state.

Resolution: No action is required. The new ″W″ state in the CONST_CHECKED column is used to indicate that the constraints type was previously checked by the user, and that some data in the table may need to be checked for integrity.

The ″N″ state does not clarify whether there exists any old data that has not yet been verified by the database manager. On a subsequent SET INTEGRITY ... IMMEDIATE CHECKED INCREMENTAL statement, the database manager must return an error, because data integrity cannot be guaranteed if only new changes have been checked. On the other hand, 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 integrity of data in the table. If the INCREMENTAL option is not specified, the database manager will choose full processing, change the ″W″ state to a ″Y″ state, and assume responsibility for maintaining data integrity.

Database Security and Tuning

Creating Databases Using Clients in DB2 Universal Database Version 6

WIN	UNIX	OS/2

Change: The method used by clients to create a database.

Symptom: Using a back-level client to create a database will result in errors.

Resolution: When using a client to create a database, ensure that the client and the server are running the same level of DB2 code.

SELECT Privilege Required on Hierarchy in DB2 Universal Database Version 6

WIN 32-bit	UNIX	OS/2

Change: Specification of the ONLY keyword (for a table) now requires that the user have SELECT privilege on all subtables of the specified typed table. Similarly, specification of the ONLY keyword (for a view) now requires that the user have SELECT privilege on all subviews of the specified typed table. Previous versions of DB2 only required SELECT privilege on the specified table or view.

Symptom: There are two possible symptoms:

- An authorization error (SQLCODE -551, SQLSTATE 42501) occurs when rebinding a package containing an SQL statement that specifies the ONLY keyword in a FROM clause, if the authorization ID under which the package was bound lacks the SELECT privilege on the subtables of the named typed table (or view).
- If the definition of a view or trigger contains the ONLY keyword 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 the SELECT privilege on all of the subtables of the named table (or view).

Resolution: The authorization ID that needs to rebind a package, or to create 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

WIN	UNIX	OS/2

Change: The following profile registry or environment variables are obsolete:

- DB2_VECTOR

Resolution: These variables are no longer needed.

Utilities and Tools

Current Explain Mode in DB2 Universal Database Version 6

WIN	UNIX	OS/2

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), the value may be truncated from 254 to 8 bytes.

Resolution: Redefine the type of all host variables that 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

WIN	UNIX	OS/2

Change: As of Version 6, the USING and SORT BUFFER parameters of the LOAD command are no longer supported. These parameters are ignored.

Symptom: A warning message is returned, stating that the USING and SORT BUFFER parameters are no longer supported, and will be ignored by the load utility.

Resolution: Ignore the warning message. For additional information, refer to the *Data Movement Utilities Guide and Reference*.

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 98, and Windows 95 (but not on Windows 3.1).

Symptom: None.

Resolution: None.

Configuration Parameters

Obsolete Database Configuration Parameters

WIN	UNIX	OS/2

Change: The following database configuration parameters are obsolete:
- DL_NUM_BACKUP (replaced by NUM_DB_BACKUP database configuration parameter)

Resolution: Remove all references to these parameters from your applications.

Appendix D. National Language Support (NLS)

This section contains information about the national language support (NLS) provided by DB2, including information about countries/regions, languages, and code pages (code sets) supported, and how to configure and use DB2 NLS features in your databases and applications.

National Language Versions

DB2 Version 7 is available in English, French, German, Italian, Spanish, Brazilian Portuguese, Japanese, Korean, Simplified Chinese, Traditional Chinese, Danish, Finnish, Norwegian, Swedish, Czech, Dutch, Hungarian, Polish, Turkish, Russian, Bulgarian, and Slovenian.

Country/Region Code and Code Page Support

Table 28 on page 222 shows the languages and code sets supported by the database servers, and how these values are mapped to country/region code and code page values that are used by the database manager.

The following is an explanation of each column in the table:

- **Code Page** shows the IBM-defined code page as mapped from the operating system code set.
- **Group** shows whether a code page is single-byte (″S″) or double-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.
- **Code Set** shows the code set associated with the supported language. The code set is mapped to the DB2 code page.
- **Tr.** shows the territory identifier.
- **Country/Region Code** shows the code that is used by the database manager internally to provide region-specific support.
- **Locale** shows the locale values supported by the database manager.
- **OS** shows the operating system that supports the languages and code sets.
- **Country/Region Name** shows the name of the region, country or countries.

Table 28. Supported Languages and Code Sets

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region 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-37	AL	355	-	HOST	Albania
1140	S-1	IBM-1140	AL	355	-	HOST	Albania
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-37	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	ISO-8859-1	AT	43	de_AT	Linux	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-37	AT	43	-	HOST	Austria
1140	S-1	IBM-1140	AT	43	-	HOST	Austria

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
915	S-5	ISO8859-5	BY	375	-	OS2	Belarus
915	S-5	ISO8859-5	BY	375	be_BY	AIX	Belarus
1131	S-5	IBM-1131	BY	375	-	OS2	Belarus
1251	S-5	1251	BY	375	-	WIN	Belarus
1283	S-5	1283	BY	375	-	Mac	Belarus
1025	S-5	IBM-1025	BY	375	-	HOST	Belarus
274	S-1	IBM-274	BE	32	-	HOST	Belgium
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	AIX	Belgium
850	S-1	IBM-850	BE	32	Nl_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	ISO-8859-1	BE	32	nl_BE	Linux	Belgium
819	S-1	ISO8859-1	BE	32	nl_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
819	S-1	ISO-8859-1	BR	55	pt_BR	Linux	Brazil
1252	S-1	1252	BR	55	-	WIN	Brazil
37	S-1	IBM-37	BR	55	-	HOST	Brazil
1140	S-1	IBM-1140	BR	55	-	HOST	Brazil

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
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
819	S-1	ISO-8859-1	CA	1	en_CA	Linux	Canada
1252	S-1	1252	CA	1	-	WIN	Canada
1275	S-1	1275	CA	1	-	Mac	Canada
37	S-1	IBM-37	CA	1	-	HOST	Canada
1140	S-1	IBM-1140	CA	1	-	HOST	Canada
863	S-1	IBM-863	CA	2	-	OS2	Canada (French)
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	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)
5488**	D-4		CN	86	-		China (PRC)

** Code page 5488 can only be used with the LOAD or IMPORT utilities to move data from code page 5488 to a DB2 Unicode database, or to EXPORT from a DB2 Unicode database to code page 5488. For more information, see the Data Movement Utilities Guide and Reference section of the Version 7.2 FixPak 4 Release Notes.

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
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
912	S-2	ISO-8859-2	HR	385	hr_HR	Linux	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

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
852	S-2	IBM-852	CZ	421	-	OS2	Czech Republic
912	S-2	ISO8859-2	CZ	421	cs_CZ	AIX	Czech Republic
912	S-2	iso88592	CZ	421	cs_CZ.iso88592	HP	Czech Republic
912	S-2	ISO8859-2	CZ	421	cs_CZ.ISO8859-2	SCO	Czech Republic
912	S-2	ISO-8859-2	CZ	421	cs_CZ	Linux	Czech Republic
1250	S-2	1250	CZ	421	-	WIN	Czech Republic
1282	S-2	1282	CZ	421	-	Mac	Czech Republic
870	S-2	IBM-870	CZ	421	-	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
819	S-1	ISO-8859-1	DK	45	da_DK	Linux	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	Denamrk
922	S-10	IBM-922	EE	372	-	OS2	Estonia
922	S-10	IBM-922	EE	372	Et_EE	AIX	Estonia
1257	S-10	1257	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
819	S-1	ISO-8859-1	FI	358	fi_FI	Linux	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

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
----	-----	--------	--	---	-----	----	--------------
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
819	S-1	ISO-8859-1	FR	33	fr_FR	Linux	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
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
819	S-1	ISO-8859-1	DE	49	de_DE	Linux	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

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
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
813	S-7	ISO-8859-7	GR	30	gr_GR	Linux	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
912	S-2	ISO-8859-2	HU	36	hu_HU	Linux	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
819	S-1	ISO-8859-1	IS	354	is_IS	Linux	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 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region 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
819	S-1	ISO-8859-1	IE	353	en_IE	Linux	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
806	S-12	IBM-806	IN	91	hi_IN	-	India
1137	S-12	IBM-1137	IN	91	-	HOST	India
862	S-8	IBM-862	IL	972	-	OS2	Israel
916	S-8	ISO8859-8	IL	972	iw_IL	AIX	Israel
856	S-8	IBM-856	IL	972	Iw_IL	AIX	Israel
916	S-8	ISO-8859-8	IL	972	iw_IL	Linux	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
819	S-1	ISO-8859-1	IT	39	it_IT	Linux	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 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region 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
943*	D-1	IBM-943	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	Sun	Japan
943	D-1	IBM-943	JP	81	ja_JP.PCK	Sun	Japan
954	D-1	EUC-JP	JP	81	ja_JP	Linux	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
1390	D-1		JP	81	-	HOST	Japan
1399	D-1		JP	81	-	HOST	Japan
1394**	D-1		JP	81	-		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	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

* On AIX 4.3 or later the code page is 943. If you are using AIX 4.2 or earlier, the code page is 932.

** Code page 1394 can only be used with the LOAD or IMPORT utilities to move data from code page 1394 to a DB2 Unicode database, or to EXPORT from a DB2 Unicode database to code page 1394. For more information, see the Data Movement Utilities Guide and Reference section of the Version 7.2 FixPak 4 Release Notes.

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
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
819	S-1	ISO-8859-1	Lat	3	-	Linux	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
1257	S-10	1257	LV	371	-	WIN	Latvia
1112	S-10	IBM-1112	LV	371	-	HOST	Latvia
921	S-10	IBM-921	LT	370	-	OS2	Lithuania
921	S-10	IBM-921	LT	370	Lt_LT	AIX	Lithuania
1257	S-10	1257	LT	370	-	WIN	Lithuania
1112	S-10	IBM-1112	LT	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
819	S-1	ISO-8859-1	NL	31	nl_NL	Linux	Netherlands
1252	S-1	1252	NL	31	-	WIN	Netherlands
1275	S-1	1275	NL	31	-	Mac	Netherlands
37	S-1	IBM-37	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-37	NZ	64	-	HOST	New Zealand
1140	S-1	IBM-1140	NZ	64	-	HOST	New Zealand

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
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
819	S-1	ISO-8859-1	NO	47	no_NO	Linux	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
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
912	S-2	ISO-8859-2	PL	48	pl_PL	Linux	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
819	S-1	ISO-8859-1	PT	351	pt_PT	Linux	Portugal
1252	S-1	1252	PT	351	-	WIN	Portugal
1275	S-1	1275	PT	351	-	Mac	Portugal
37	S-1	IBM-37	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
912	S-2	ISO-8859-2	RO	40	ro_RO	Linux	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

Table 28. Supported Languages and Code Sets *(continued)*

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
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
915	S-5	ISO-8859-5	RU	7	ru_RU	Linux	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
852	S-2	IBM-852	SK	422	-	OS2	Slovakia
912	S-2	ISO8859-2	SK	422	sk_SK	AIX	Slovakia
912	S-2	iso88592	SK	422	sk_SK.iso88592	HP	Slovakia
912	S-2	ISO8859-2	SK	422	sk_SK.ISO8859-2	SCO	Slovakia
1250	S-2	1250	SK	422	-	WIN	Slovakia
1282	S-2	1282	SK	422	-	Mac	Slovakia
870	S-2	IBM-870	SK	422	-	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
912	S-2	ISO-8859-2	SI	386	sl_SI	Linux	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

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
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
850	S-1	IBM-850	ES	34	Es_ES	AIX	Spain
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
819	S-1	ISO-8859-1	ES	34	es_ES	Linux	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
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
819	S-1	ISO-8859-1	SE	46	sv_SE	Linux	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
819	S-1	ISO-8859-1	CH	41	de_CH	Linux	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

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
----	-----	--------	--	---	-----	----	--------------
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	Sun	Taiwan
964	D-2	cns11643	TW	88	zh_TW	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
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
920	S-9	ISO-8859-9	TR	90	tr_TR	Linux	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.
819	S-1	ISO-8859-1	GB	44	en_GB	Linux	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.

Table 28. Supported Languages and Code Sets (continued)

Code Page	Group	Code-Set	Tr.	Country/ Region Code	Locale	OS	Country/ Region Name
----	-----	--------	--	---	-----	----	--------------
1125	S-5	IBM-1125	UA	380	-	OS2	Ukraine
1124	S-5	IBM-1124	UA	380	uk_UA	AIX	Ukraine
1251	S-5	1251	UA	380	-	WIN	Ukraine
1123	S-5	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
819	S-1	ISO-8859-1	US	1	en_US	Linux	USA
1252	S-1	1252	US	1	-	WIN	USA
1275	S-1	1275	US	1	-	Mac	USA
37	S-1	IBM-37	US	1	-	HOST	USA
1140	S-1	IBM-1140	US	1	-	HOST	USA
1163	S-11	IBM-1163	VN	84	-	OS2	Vietnam
1163	S-11	IBM-1163	VN	84	vi_VN	AIX	Vietnam
1258	S-11	1258	VN	84	-	WIN	Vietnam
1164	S-11	IBM-1164	VN	84	-	HOST	Vietnam

```
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) */
```

Table 28. Supported Languages and Code Sets (continued)

```
                               Country/
Code                            Region                                   Country/
Page   Group  Code-Set  Tr.  Code Locale           OS         Region Name
----   -----  --------  --   ---  -----           ----      --------------
```

```
The following map to English (US):
---------------------------------
    /* English (Jamaica) */
    /* English (Caribbean) */
```

```
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) */
```

Notes:

1. 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

2. 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 want to continue to use the previous definition of these code pages, you can 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 back up the existing `IBM01252.usc` and `IBM00850.ucs` files before copying the non-Euro versions over them. After copying the files, you will not have Euro currency symbol support from DB2 UDB.

Locale Setting for the DB2 Administration Server

Ensure that the locale of the DB2 Administration Server instance is compatible with the locale of the DB2 instance. Otherwise, the DB2 instance cannot communicate with the DB2 Administration Server.

If the LANG environment variable is not set in the user profile of the DB2 Administration Server, the DB2 Administration Server will be started with the default system locale. If the default system locale is not defined, the DB2 Administration Server will be started with code page 819. If the DB2 instance uses one of the DBCS locales, and the DB2 Administration Server is started with code page 819, the instance will not be able to communicate with the DB2 Administration Server. The locale of the DB2 Administration Server and the locale of the DB2 instance must be compatible.

For example, on a Simplified Chinese Linux system, `LANG=zh_CN` should be set in the DB2 Administration Server's user profile.

Supported Translated Locales for UNIX-based Platforms

On UNIX-based platforms, the DB2 product messages and library can be installed in several different languages. The DB2 installation utility lays down the message catalog file sets into the most commonly used locale directory for a given platform as shown in the following tables. Table 29 on page 238 provides information for AIX, HP-UX, and Solaris. Table 30 on page 239 provides information for Linux, Linux/390, SGI, and Dynix.

If your system uses the same code pages but different locale names than those provided in Table 29 and Table 30 on page 239, you can still see the translated messages by creating a link to the appropriate message directory.

For example, if your AIX machine default locale is `ja_JP.IBM-eucJP` and the code page of `ja_JP.IBM-eucJP` is 954, you can create a link from `/usr/lpp/db2_07_01/msg/ja_JP.IBM-eucJP` to `/usr/lpp/db2_07_01/msg/ja_JP` by issuing the following command:

ln -s /usr/lpp/db2_07_01/msg/ja_JP /usr/lpp/db2_07_01/msg/ja_JP.IBM-eucJP

After the execution of this command, all DB2 messages come up in Japanese.

Table 29. Directory Locales for AIX, HP-UX, Solaris

	AIX		**HP-UX**		**Solaris**	
Language	**Locale**	**Code Page**	**Locale**	**Code Page**	**Locale**	**Code Page**
French	fr_FR Fr_FR	819 850	fr_FR.iso88591 fr_FR.roman8	819 1051	fr	819
German	de_DE De_DE	819 850	de_DE.iso88591 de_DE.roman8	819 1051	de	819
Italian	it_IT It_IT	819 850	it_IT.iso88591 it_IT.roman8	819 1051	it	819
Spanish	es_ES Es_ES	819 850	es_ES.iso88591 es_ES.roman8	819 1051	es	819
Brazilian Portuguese	pt_BR	819			pt_BR	819
Japanese	ja_JP Ja_JP	954 932	ja_JP.eucJP	954	ja	954
Korean	ko_KR	970	ko_KR.eucKR	970	ko	970
Simplified Chinese	zh_CN Zh_ CN.GBK	1383 1386	zh_CN.hp15CN	1383	zh	1383
Traditional Chinese	zh_TW Zh_TW	964 950	zh_TW.eucTW zh_TW.big5	964 950	zh_TW zh_TW.BIG5	964 950
Danish	da_DK Da_DK	819 850	da_DK.iso88591 da_DK.roman8	819 1051	da	819

Table 29. Directory Locales for AIX, HP-UX, Solaris (continued)

	AIX		**HP-UX**		**Solaris**	
Language	**Locale**	**Code Page**	**Locale**	**Code Page**	**Locale**	**Code Page**
Finnish	fi_FI Fi_FI	819 850	fi_FI.iso88591 fi_FI.roman8	819 1051	fi	819
Norwegian	no_NO No_NO	819 850	no_NO.iso88591 no_NO.roman8	819 1051	no	819
Sweden	sv_SE Sv_SE	819 850	sv_SE.iso88591 sv_SE.roman8	819 1051	sv	819
Czech	cs_CZ	912				
Hungarian	hu_HU	912				
Polish	pl_PL	912				
Dutch	nl_NL Nl_NL	819 850				
Turkish	tr_TR	920				
Russian	ru_RU	915				
Bulgarian	bg_BG	915	bg_BG.iso88595	915		
Slovenian	sl_SI	912	sl_SI.iso88592	912	sl_SI	912

Table 30. Directory Locales for Linux, Linux/390, SGI, Dynix

	Linux		**Linux/390**		**SGI**		**Dynix**	
Language	**Locale**	**Code Page**	**Locale**	**Code Page**	**Locale**	**Code Page**	**Locale**	**Code Page**
French	fr	819	fr	819			fr	819
German	de	819	de	819			de	819
Italian							es	819
Spanish								
Brazilian Portuguese								
Japanese	ja_JP.ujis	954	ja_JP.ujis	954			ja_JP.EUC	954
Korean	ko	970	ko	970	ko_KO.euc	970		
Simplified Chinese	zh zh_CN.GBK	1386	zh zh_CN.GBK	1386				

Table 30. Directory Locales for Linux, Linux/390, SGI, Dynix (continued)

	Linux		**Linux/390**		**SGI**		**Dynix**	
Language	**Locale**	**Code Page**	**Locale**	**Code Page**	**Locale**	**Code Page**	**Locale**	**Code Page**
Traditional Chinese	zh_TW.Big5	950	zh_TW.Big5	950				
Danish								
Finnish								
Norwegian								
Sweden								
Czech								
Hungarian								
Polish								
Dutch							nl	819
Turkish								
Russian								
Bulgarian								
Slovenian								

Control Center and Documentation File Sets

The Control Center, Control Center Help and documentation file sets are placed in the following directories on the target workstation:

- DB2 for AIX:
  - `/usr/lpp/db2_07_01/cc/%L`
  - `/usr/lpp/db2_07_01/java/%L`
  - `/usr/lpp/db2_07_01/doc/%L`
  - `/usr/lpp/db2_07_01/qp/%L`
  - `/usr/lpp/db2_07_01/spb/%L`
- DB2 for HP-UX:
  - `/opt/IBMdb2/V7.1/cc/%L`
  - `/opt/IBMdb2/V7.1/java/%L`
  - `/opt/IBMdb2/V7.1/doc/%L`
- DB2 for Linux:
  - `/usr/IBMdb2/V7.1/cc/%L`
  - `/usr/IBMdb2/V7.1/java/%L`
  - `/usr/IBMdb2/V7.1/doc/%L`
- DB2 for Solaris:

- /opt/IBMdb2/V7.1/cc/%L
- /usr/IBMdb2/V7.1/java/%L
- /opt/IBMdb2/V7.1/doc/%L

Control Center file sets are in Unicode code page. Documentation and Control Center help file sets are in a browser-recognized code set. If your system uses a different locale name than the one provided, you can still run the translated version of the Control Center and see the translated version of Help by creating links to the appropriate language directories.

For example, if your AIX machine default locale is `ja_JP.IBM-eucJP`, you can create links from `/usr/lpp/db2_07_01/cc/ja_JP.IBM-eucJP` to `/usr/lpp/db2_07_01/cc/ja_JP` and from `/usr/lpp/db2_07_01/doc/ja_JP.IBM-eucJP` to `/usr/lpp/db2_07_01/doc/ja_JP` by issuing the following commands:

- **ln -s /usr/lpp/db2_07_01/cc/ja_JP /usr/lpp/db2_07_01/cc/ja_JP.IBM-eucJP**
- **ln -s /usr/lpp/db2_07_01/doc/ja_JP /usr/lpp/db2_07_01/doc/ja_JP.IBM-eucJP**

After the execution of these commands, the Control Center and Help come up in Japanese.

Note: The Web Control Center is not supported running natively on Linux/390 or NUMA-Q. It can be used from a client workstation to manage databases on these platforms.

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. Normally, you do not need to set the DB2CODEPAGE registry variable, because DB2 automatically derives the code page information from the operating system. Setting the DB2CODEPAGE registry variable to incorrect values may cause unpredictable results.

In the case of applications coded to use the CLI interface, the CLI code layer will use the locale settings in some cases, even if the user has set the DB2CODEPAGE registry variable.

The *database code page* is derived from the value specified (explicitly or by default) at the time the database is created. For example, the following defines how the *active environment* is determined in different operating environments:

UNIX	On UNIX based operating systems, the active environment is determined from the locale setting, which includes information about language, territory and code set.

OS/2	On 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.
Macintosh	For the Macintosh operating system, if the DB2CODEPAGE registry variable is not set, the Macintosh code page is derived from the Regional version code from the installed script.
Windows	For all Windows 32-bit operating systems, if the DB2CODEPAGE registry 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 28 on page 222.

Character Sets

| The database manager does not, in general, restrict the character set available to an application. For a detailed explanation of multibyte character sets (MBCS) supported by DB2, refer to the *Application Development Guide*.

Character Set for Identifiers

| The basic character set that can 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 is augmented with three special characters (#, @, and $) to provide compatibility with host database products. Use special characters #, @, and $ with care in an NLS environment because they are not included in the NLS host (EBCDIC) invariant character set. Characters from the extended character set can also be used, depending on the code page that is being used. If you are using the database in a multiple code page environment, you must ensure that all code pages support any elements from the extended character set you plan to use.

When naming database objects (such as tables and views), program labels, host variables, cursors, and elements from the extended character set (for example, letters with diacritical marks) can also be used. Precisely which characters are available depends on the code page in use. If you are using the database in a multiple code page environment, you must ensure that all code pages support any elements from the extended character set that you plan to use. For information about delimited identifiers that have characters outside of the extended character set, but which can be used in SQL statements, refer to the *SQL Reference*.

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 the following:

- 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 SQL Statements

| The coding of SQL statements is not language dependent. SQL keywords can
| be typed in uppercase, lowercase, or mixed case. The names of database
| objects and host variables, as well as program labels in an SQL statement
| cannot contain characters that are outside of the extended character set
| described in “Extended Character Set Definition for DBCS Identifiers”.

Bidirectional CCSID Support

The following BiDi attributes are required for correct handling of bidirectional data on different platforms:

```
- Text type (LOGICAL or VISUAL)
- Shaping (SHAPED or UNSHAPED)
- Orientation (RIGHT-TO-LEFT or LEFT-TO-RIGHT)
- Numeral shape (ARABIC or HINDI)
- Symmetric swapping (YES or NO)
```

Because default values on different platforms are not the same, problems can occur when DB2 data is moved from one platform to another. For example, the Windows operating system uses LOGICAL UNSHAPED data, while OS/390 usually uses SHAPED VISUAL data. Therefore, without support for bidirectional attributes, data sent from DB2 Universal Database for OS/390 to DB2 UDB on 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 (dec)	CCSID (hex)	Code Page	String Type
00420	x'01A4'	420	4
00424	x'01A8'	424	4
08612	x'21A4'	420	5
08616	x'21A8'	424	10
00856	x'0358'	856	5
00862	x'035E'	862	4
00864	x'0360'	864	5
00916	x'0394'	916	5
01046	x'0416'	1046	5
01089	x'0441'	1089	5
01255	x'04E7'	1255	5
01256	x'04E8'	1256	5
62208	x'F300'	856	4
62209	x'F301'	862	10
62210	x'F302'	916	4
62211	x'F303'	424	5
62213	x'F305'	862	5
62215	x'F307'	1255	4
62218	x'F30A'	864	4
62220	x'F30C'	856	6
62221	x'F30D'	862	6
62222	x'F30E'	916	6
62223	x'F30F'	1255	6
62224	x'F310'	420	6
62225	x'F311'	864	6
62226	x'F312'	1046	6
62227	x'F313'	1089	6
62228	x'F314'	1256	6
62229	x'F315'	424	8
62230	x'F316'	856	8
62231	x'F317'	862	8
62232	x'F318'	916	8
62233	x'F319'	420	8
62234	x'F31A'	420	9
62235	x'F31B'	424	6
62236	x'F31C'	856	10
62237	x'F31D'	1255	8
62238	x'F31E'	916	10
62239	x'F31F'	1255	10
62240	x'F320'	424	11
62241	x'F321'	856	11
62242	x'F322'	862	11
62243	x'F323'	916	11
62244	x'F324'	1255	11
62245	x'F325'	424	10
62246	x'F326'	1046	8

62247	x'F327'	1046	9
62248	x'F328'	1046	4
62249	x'F329'	1046	12
62250	x'F32A'	420	12

where CDRA string types are defined as:

String Type	Text Type	Numerical Shape	Orientation	Shaping	Symmetrical Swapping
4	Visual	Passthru	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	Passthru	Contextual-L	Unshaped	ON
11	Implicit	Passthru	Contextual-R	Unshaped	ON
12	Implicit	Arabic	RTL	Shaped	ON

Note: (*) Field orientation is left-to-right (LTR) when the first alphabetic character is a Latin character, and right-to-left (RTL) when it is a bidirectional (RTL) character. Characters are unshaped, but LamAlef ligatures are kept, and are 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. It 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 particular bidirectional CCSID in a non-DRDA environment, select the 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 then reconnect to allow the new setting for DB2CODEPAGE to take effect. If you select a CCSID that is not appropriate for the code page or string type of your client platform, you may get unexpected results. If you select an incompatible CCSID (for example, the Hebrew CCSID for connection to an Arabic database), 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 only need to set the DB2CODEPAGE value. However, if the HOST platform does not support these CCSIDs, you must also specify a CCSID override for the HOST database server to which you are connecting. This is necessary because, in a DRDA environment, code page conversions and layout transformations are performed by the receiver of data. However, if the 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 information about setting a CCSID override, refer to the *DB2 Connect User's Guide*.

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 the server, it is usually the receiver that performs conversion on the incoming data. The same convention would normally apply to bidirectional layout transformations, and 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 CCSID of the server database must 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 want 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 must still add the BIDI parameter to the PARMS field of the DCS database directory. 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 you want to override the default server database bidirectional CCSID:

```
",,,,,,,,BIDI=xyz"
```

where *xyz* is the CCSID override.

Note: The registry variable DB2BIDI must be set to YES for the BIDI parameter to take effect.

A list of the bidirectional CCSIDs that are supported, along with their string types, can be found in “Bidirectional-Specific CCSIDs” on page 243.

The use of this feature is best described with an example.

Suppose you have a Hebrew DB2 client running CCSID 62213 (bidirectional string type 5), and you want 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 based on CCSID 08616 (bidirectional string type 6).

There are two problems here: 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 (62213). It only supports CCSID 00862, which is based on the same code page as CCSID 62213.

You will need to ensure 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"
```

This command tells 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. This 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 in such a way 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; use host variables whenever possible.

If a specific bidirectional CCSID is causing problems that cannot be rectified by following these recommendations, set DB2BIDI to 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. 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 collating sequence is specified at database creation time, and cannot be modified later.

The database manager allows databases to be created with custom collating sequences, using the application programming interface (API). For information about implementing a custom collating sequence table, refer to the *Application Development Guide*.

Note: Character string data defined with the FOR BIT DATA attribute, and BLOB data, is sorted using the binary sort sequence.

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 about how the database collating sequence is used, see "String Comparisons" in the *SQL Reference*.

Potential problems can 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 may or may 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 character code points will only match query results that are 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 push down order-dependent processing involving character data to the data source. If a data source collating sequence does not match the DB2 collating sequence, data is retrieved, and all order-dependent processing on character data is done locally (this can reduce performance).

To determine whether a data source and DB2 have the same collating sequence, consider the following:

- 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 results that are different 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 the identity sequence at database creation time (Oracle databases use an equivalent collating sequence).
- If a DB2 database will access only DB2 UDB databases, ensure that you match collating sequence values.

For information about setting up an MVS collating sequence, refer to the *Administrative API Reference* for examples under the description of the **sqlecrea** - Create Database API. These examples contain collation tables for the 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. This option specifies whether 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 may receive incorrect results. For example, if your plan uses merge joins, the DB2 optimizer will push down ordering operations to the data sources as much as possible. If the data source collating sequence is not the same, the join result set may not be correct.

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 whether the collating sequence at the data source is identical to the DB2 collating sequence.

| **Sorting Thai Characters**
| If you want to sort special vowels (″leading vowels″), tonal marks and other
| special Thai characters correctly, you must create the database with the
| COLLATE USING NLSCHAR clause in the CREATE DATABASE command.
| See the *Command Reference* for the syntax.

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. The range of the other parts is 0 to 59. If the hour is 24, the minute and second specifications are 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.

Time Stamp

A *time stamp* is a seven-part value (year, month, day, hour, minute, second, and microsecond) designating a date and a time of day as defined above, except that the time includes the specification of microseconds.

The internal representation of a time stamp is a string of 10 bytes. Each byte is 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 a 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 time stamps can also, however, be represented by character strings, and these representations directly concern the SQL user, because 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/region code of the client, unless overridden by specification of the ″F″ format option when the program is precompiled or bound to the database. For a list of the string formats for the various country/region codes, see Table 33 on page 254.

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 time stamp before the operation is performed. Valid string representations of datetime values are defined in the following sections.

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 part and the day part of the date.

Valid string formats for dates are listed in Table 31. Each format is identified by name, and includes an associated abbreviation and an example of its use.

Table 31. 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

Table 31. Formats for String Representations of Dates (continued)

Format Name	Abbreviation	Date Format	Example
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/region 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 32. Each format is identified by name, and includes an associated abbreviation and an example of its use.

Table 32. Formats for String Representations of Times

Format Name	Abbreviation	Time Format	Example
International Standards Organization	ISO	hh.mm.ss	13.30.05
IBM USA standard	USA	hh:mm AM or PM	1:30 PM
IBM European standard	EUR	hh.mm.ss	13.30.05
Japanese Industrial Standard Christian Era	JIS	hh:mm:ss	13:30:05
Site-defined (Local)	LOC	Depends on application country/region code	—

Notes:

1. In ISO, EUR, or JIS time string format, `.ss` (or `:ss`) is optional.
2. In USA time string format, the minutes specification can be omitted, indicating an implicit specification of 00 minutes. Thus, 1 PM is equivalent to 1:00 PM.

3. In USA time string format, the hours specification cannot be greater than 12, and cannot be 0, except in 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.

Time Stamp Strings

A string representation of a time stamp is a character string that starts with a digit and has a length of at least 16 characters. The complete string representation of a time stamp has the form *yyyy-mm-dd-hh.mm.ss.nnnnnn*. Trailing blanks may be included; leading zeros may be omitted from the month, day, or hour part of the time stamp, and microseconds may be truncated or omitted entirely. If you choose to omit any digit of the microseconds part, an implicit specification of 0 is assumed. Thus, `1991-3-2-8.30.00` is equivalent to `1991-03-02-08.30.00.000000`.

Character Set Considerations

Date and time stamp 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/region code of the application. This default format can be overridden by specifying the ″F″ format option when the program is precompiled or bound to the database.

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/region 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 33 on page 254.
- Output Date Format

– The default output date format is shown in Table 33.

Note: Table 33 also shows a listing of the string formats for the various country/region codes.

Table 33. Date and Time Formats by Country/Region Code

Country/Region Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
355 Albania	yyyy-mm-dd	JIS	LOC	LOC, USA, EUR, ISO
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.yyyy	JIS	LOC	LOC, EUR, ISO
359 Bulgaria	dd.mm.yyyy	JIS	EUR	LOC, USA, EUR, ISO
001 Canada	mm-dd-yyyy	JIS	USA	LOC, USA, EUR, ISO
002 Canada (French)	dd-mm-yyyy	ISO	ISO	LOC, USA, EUR, ISO
385 Croatia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
042 Czech Republic	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
045 Denmark	dd-mm-yyyy	ISO	ISO	LOC, USA, EUR, ISO
358 Finland	dd/mm/yyyy	ISO	EUR	LOC, EUR, ISO
389 FYR Macedonia	dd.mm.yyyy	JIS	EUR	LOC, USA, EUR, ISO
033 France	dd/mm/yyyy	JIS	EUR	LOC, EUR, ISO
049 Germany	dd/mm/yyyy	ISO	ISO	LOC, EUR, ISO
030 Greece	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
036 Hungary	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
354 Iceland	dd-mm-yyyy	JIS	LOC	LOC, USA, EUR, ISO
091 India	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO

Table 33. Date and Time Formats by Country/Region Code (continued)

Country/Region Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
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 People's Republic of China	mm/dd/yyyy	JIS	ISO	LOC, USA, EUR, ISO
040 Romania	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
007 Russia	dd/mm/yyyy	ISO	LOC	LOC, EUR, ISO
381 Serbia/ Montenegro	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
042 Slovakia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
386 Slovenia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
034 Spain	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
046 Sweden	dd/mm/yyyy	ISO	ISO	LOC, EUR, ISO
041 Switzerland	dd/mm/yyyy	ISO	EUR	LOC, EUR, ISO
088 Taiwan	mm-dd-yyyy	JIS	ISO	LOC, USA, EUR, ISO
066 Thailand (2)	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
090 Turkey	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO

Table 33. Date and Time Formats by Country/Region Code (continued)

Country/Region Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
044 UK	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
001 USA	mm-dd-yyyy	JIS	USA	LOC, USA, EUR, ISO
084 Vietnam	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO

Notes:

1. Countries/Regions using the default C locale are assigned country/region code 001.
2. yyyy in Buddhist era is equivalent to Gregorian + 543 years (Thailand only).

Unicode Support in DB2 UDB

The level of Unicode support in DB2 UDB is documented here.

Introduction

The Unicode character encoding standard is a fixed-length, character encoding scheme that includes characters from almost all of the living languages of the world.

Unicode uses two encoding forms: 8-bit and 16-bit. The default encoding form is 16-bit, that is, each character is 16 bits (two bytes) wide, and is usually shown as U+hhhh, where hhhh is the hexadecimal code point of the character. 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 the encoding of as many as one million more characters. The extension mechanism uses a pair of high and low surrogate characters to encode one extended character. The first (or high) surrogate character has a code value between U+D800 and U+DBFF, and the second (or low) surrogate character has a code value between U+DC00 and U+DFFF.

The International Standards Organization (ISO) and the International Electrotechnical Commission (IEC) standard 10646 (ISO/IEC 10646) specifies the Universal Multiple-Octet Coded Character Set (UCS) that has a 16-bit (two-byte) version (UCS-2) and a 32-bit (four-byte) version (UCS-4). UCS-2 is identical to the Unicode 16-bit form without surrogates. ISO/IEC 10646 also defines an extension technique for encoding some UCS-4 characters using two UCS-2 characters. This extension, called UTF-16, is identical to the Unicode 16-bit encoding form with surrogates. In summary, the UTF-16 character repertoire consists of all the UCS-2 characters plus the additional one million characters accessible via the surrogate pairs.

When serializing 16-bit Unicode characters into bytes, some processors place the most significant byte in the initial position (known as big-endian order), while others place the least significant byte first (known as little-endian order). The default byte ordering for Unicode is big-endian.

More information on Unicode can be found in the latest edition of The Unicode Standard book, and from The Unicode Consortium web site (www.unicode.org).

UTF-8

Sixteen-bit Unicode characters pose a major problem for byte-oriented ASCII-based applications and file systems. For example, non-Unicode aware applications may misinterpret the leading 8 zero bits of the uppercase character 'A' (U+0041) as the single-byte ASCII NULL character.

UTF-8 (UCS Transformation Format 8) is an algorithmic transformation that transforms fixed-length Unicode characters into variable-length ASCII-safe byte strings. In UTF-8, ASCII and control characters are represented by their usual single-byte codes, and other characters become two or more bytes long. The number of bytes for each UTF-16 character in UTF-8 format can be determined from Table 34.

Table 34. UTF-8 Bit Distribution

Code Value (binary)	**UTF-16 (binary)**	**1st byte (binary)**	**2nd byte (binary)**	**3rd byte (binary)**	**4th byte (binary)**
00000000 0xxxxxxx	00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
uuuuu zzzzyyyy yyxxxxxx	110110ww wwzzzzyy 110111yy yyxxxxxx	11110uuu (where uuuuu = wwww+1)	10uuzzzz	10yyyyyy	10xxxxxx

In each of the above, the series of u's, w's, x's, y's, and z's is the bit representation of the character. For example, U+0080 transforms into 11000010

10000000 in binary, and the surrogate character pair U+D800 U+DC00 becomes 11110000 10010000 10000000 10000000 in binary.

Unicode Implementation in DB2 UDB

DB2 UDB supports UCS-2; that is, Unicode without surrogates.

In versions of DB2 UDB prior to Version 7.2 FixPak 4, DB2 UDB treats the two characters in a surrogate pair as two independent Unicode characters. Therefore transforming the pair from UTF-16/UCS-2 to UTF-8 results in two three-byte sequences (see the second last row in Table 34 on page 257). Starting in DB2 UDB Version 7.2 FixPak 4, DB2 UDB recognizes surrogate pairs when transforming between UTF-16/UCS-2 and UTF-8, thus a pair of UTF-16 surrogates will become one UTF-8 four-byte sequence (see the last row in Table 34 on page 257).

DB2 UDB treats each Unicode character, including those (non-spacing) characters such as the COMBINING ACUTE ACCENT character (U+0301), as an individual character. Therefore DB2 UDB would not recognize that the character LATIN SMALL LETTER A WITH ACUTE (U+00E1) is canonically equivalent to the character LATIN SMALL LETTER A (U+0061) followed by the character COMBINING ACUTE ACCENT (U+0301).

Code Page/CCSID Numbers

Within IBM, the UCS-2 code page has been registered as code page 1200, with a growing character set; 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.

A specific version 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 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 same (superset) tables are used for both.

Within 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.

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 Unicode database is created, 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.

Creating a Unicode Database

By default, databases are created in the code page of the application creating them. Therefore, if you create your database from a Unicode (UTF-8) client (for example, the UNIVERSAL locale of AIX or if the DB2CODEPAGE registry variable on the client is set to 1208), your database will be created as a Unicode database. Alternatively, you can explicitly specify ″UTF-8″ as the CODESET name, and use any valid TERRITORY code supported by DB2 UDB.

For example, to create a Unicode database with the territory code for the United States of America, issue:

```
DB2 CREATE DATABASE dbname USING CODESET UTF-8 TERRITORY US
```

To create a Unicode database using the **sqlecrea** API, you should set the values in *sqledbcountryinfo* accordingly. For example, set SQLDBCODESET to `UTF-8`, and SQLDBLOCALE to any valid territory code (for example, `US`).

The default collating sequence for a UCS-2 Unicode database is IDENTITY, which orders the characters by their code points. Therefore, by default, all Unicode characters are ordered and compared according to their code points. For most Unicode characters, their binary collation orders when encoded in UTF-8 and UCS-2 are the same. But if you have any extended character that requires a pair of surrogate characters to encode, then in UTF-8 encoding the character will be collated towards the end, but in UCS-2 encoding the same character will be collated somewhere in the middle, and its two surrogate characters can be separated. The reason is the extended character, when encoded in UTF-8, has a four-byte binary code value of 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx, which is greater than the UTF-8 encoding of U+FFFF. But in UCS-2, the same extended character is encoded as two UCS-2 surrogate characters, and DB2 UDB treats and collates each surrogate character independently.

All culturally-sensitive parameters, such as date or time format, decimal separator, and others, are based on the current territory of the client.

A Unicode database allows connection from every code page supported by DB2 UDB. Code page character conversions between the 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 method, 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 may be 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 table in "UTF-8" on page 257.

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 a 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 multibyte UTF-8 encoding, each ASCII character is one byte, but non-ASCII characters take two to four 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*/4 to *n* characters.

Using character string UTF-8 encoding versus the graphic string UCS-2 data type also has an impact on the total storage requirements. In 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, in situations where the majority of characters are non-ASCII characters that expand to three- or four-byte UTF-8 sequences (for example ideographic characters), the UCS-2 graphic-string format may be a better alternative, because every three-byte UTF-8 sequence becomes a 16-bit UCS-2 character, while each four-byte UTF-8 sequence becomes two 16-bit UCS-2 characters.

In MBCS environments, SQL scalar functions that operate on character strings, such as LENGTH, SUBSTR, POSSTR, MAX, MIN, and the like, operate on the 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 UCS-2 database, because these values are always defined in the context of the database code page. That is, in the case of a UCS-2 database, these offsets 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, SQLCODE -191 (SQLSTATE 22504) is returned. For a description of the behavior of these functions, refer to the *SQL Reference*.

SQL CHAR data types are supported (in the C language) by the `char` data type in user programs. SQL GRAPHIC data types are supported by `sqldbchar` in user 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 multibyte 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 multibyte character) is allowed by DB2 UDB. For details about which identifiers allow the use of extended characters, see “Appendix A. Naming Rules” on page 187.

Clients can enter any character that is supported by their environment, and all the characters in the identifiers will be converted to UTF-8 by the database manager. Two points must be taken into account when specifying national language characters in identifiers for a UCS-2 database:

- Each non-ASCII character requires two to four bytes. Therefore, an *n*-byte identifier can only hold somewhere between *n*/4 and *n* characters, depending on 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 that is completely non-ASCII (for example, in Japanese), only *n*/4 to *n*/3 characters can be used.
- 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.

Unicode Literals

Unicode literals can be specified in two ways:

- As a graphic string constant, using the G'...' or N'....' format described in the ″Graphic String Constants″ section of the ″Language Elements″ chapter in the *SQL Reference*. Any literal specified in this way will be converted by the database manager from the application code page to 16-bit Unicode.
- As a Unicode hexadecimal string, using the UX'....' or GX'....' format. The constant specified between the quotation marks after UX or GX must be a multiple of four hexadecimal digits in big-endian order. Each four-digit group represents one 16-bit Unicode code point. Note that surrogate characters always appear in pairs, therefore you need two four-digit groups to represent the high and low surrogate characters.

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, when entering any code page 850 character from a terminal that is using code page 850). The second method should be used for characters that are outside of the application code page repertoire (for example, when 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 non-SBCS 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 non-SBCS characters.

In a Unicode database, there is really no distinction between ″single-byte″ and ″double-byte″ characters; every 16-bit character occupies two bytes. Although the UTF-8 format is a ″mixed-byte″ encoding of Unicode characters, there is no real distinction between SBCS and MBCS characters in UTF-8. Every character is a Unicode character, regardless of the number of its bytes that are in UTF-8 format. When specifying a character string, or a graphic string expression, an underscore refers to one Unicode character, and a percent sign refers to a string of zero or more Unicode characters. Note that you need two underscores to match a surrogate pair.

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 a DBCS percent sign or a DBCS underscore, but every supported code page contains a single-byte percent sign (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 sign (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 also 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 sign characters, only ASCII characters, or characters that expand into a two-byte UTF-8 sequence, are supported. If you specify an escape character that expands to a three-byte UTF-8 value, an error message (error SQL0130N, SQLSTATE 22019) is returned.

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 expected behavior for the export of any database, and cannot be changed, because the entire delimited ASCII file can have only one code page. Therefore, if you export to a delimited ASCII file, only those UCS-2 characters that exist in your application code page will 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 change 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 those clients. The range of valid
| delimiters for UTF-8 clients is X′01′ to X′7F′, with the usual restrictions. For a

complete list of these restrictions, refer to the ″Export/Import/Load Utility File Formats″ appendix in the *Data Movement Utilities Guide and Reference*.

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 a 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).

The load utility places the data directly into the database and, by default, assumes data in ASC or DEL files to be in the code page of the database. Therefore, by default, no code page conversion takes place for ASCII files. When the code page for the data file has been explicitly specified (using the `codepage` modifier), the load utility uses this information to convert from the specified code page to the database code page before loading the data. For PC/IXF files, the load utility always converts from the code pages specified in the IXF header to the database code page (1208 for CHAR, and 1200 for GRAPHIC).

The code page for DBCLOB files is always 1200 for UCS-2. The code page for CLOB files is the same as the code page for the data files being imported, loaded or exported. For example, when loading or importing data using the 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, the load utility assumes that CLOB data is in the code page of the database (unless explicitly specified otherwise using the `codepage` modifier), while the import utility assumes it to be in the code page of the client application.

The `nochecklengths` modifier is always specified for a UCS-2 database, because:

- Any SBCS can be connected to a database for which there is no DBCS code page
- Character strings in UTF-8 format usually have different lengths than those in client code pages.

For more information about the load, import, and export utilities, refer to the *Data Movement Utilities Guide and Reference*.

Incompatibilities

For applications connected to a UCS-2 database, 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 DBCS code page of the application, 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, the graphic string data is in code page 301. For the 932 client applications connected to a UCS-2 database, the graphic string data is in UCS-2.

Appendix E. Using the DB2 Library

The DB2 Universal Database library consists of online help, books (PDF and HTML), and sample programs in HTML format. This section describes the information that is provided, and how you can access it.

To access product information online, you can use the Information Center. For more information, see "Accessing Information with the Information Center" on page 281. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web.

DB2 PDF Files and Printed Books

DB2 Information

The following table divides the DB2 books into four categories:

DB2 Guide and Reference Information
These books contain the common DB2 information for all platforms.

DB2 Installation and Configuration Information
These books are for DB2 on a specific platform. For example, there are separate *Quick Beginnings* books for DB2 on OS/2, Windows, and UNIX-based platforms.

Cross-platform sample programs in HTML
These samples are the HTML version of the sample programs that are installed with the Application Development Client. The samples are for informational purposes and do not replace the actual programs.

Release notes
These files contain late-breaking information that could not be included in the DB2 books.

The installation manuals, release notes, and tutorials are viewable in HTML directly from the product CD-ROM. Most books are available in HTML on the product CD-ROM for viewing and in Adobe Acrobat (PDF) format on the DB2 publications CD-ROM for viewing and printing. You can also order a printed copy from IBM; see "Ordering the Printed Books" on page 277. The following table lists books that can be ordered.

On OS/2 and Windows platforms, you can install the HTML files under the `sqllib\doc\html` directory. DB2 information is translated into different languages; however, all the information is not translated into every language. Whenever information is not available in a specific language, the English information is provided

On UNIX platforms, you can install multiple language versions of the HTML files under the `doc/%L/html` directories, where *%L* represents the locale. For more information, refer to the appropriate *Quick Beginnings* book.

You can obtain DB2 books and access information in a variety of ways:
- "Viewing Information Online" on page 280
- "Searching Information Online" on page 284
- "Ordering the Printed Books" on page 277
- "Printing the PDF Books" on page 276

Table 35. DB2 Information

Name	**Description**	**Form Number** **PDF File Name**	**HTML Directory**
	DB2 Guide and Reference Information		
Administration Guide	*Administration Guide: Planning* provides an overview of database concepts, information about design issues (such as logical and physical database design), and a discussion of high availability. *Administration Guide: Implementation* provides information on implementation issues such as implementing your design, accessing databases, auditing, backup and recovery. *Administration Guide: Performance* provides information on database environment and application performance evaluation and tuning. You can order the three volumes of the *Administration Guide* in the English language in North America using the form number SBOF-8934.	SC09-2946 db2d1x70 SC09-2944 db2d2x70 SC09-2945 db2d3x70	db2d0
Administrative API Reference	Describes the DB2 application programming interfaces (APIs) and data structures that you can use to manage your databases. This book also explains how to call APIs from your applications.	SC09-2947 db2b0x70	db2b0

Table 35. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
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.	SC09-2948 db2axx70	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. Available in HTML format only.	No form number db2apx70	db2ap
Application Development Guide	Explains how to develop applications that access DB2 databases using embedded SQL or Java (JDBC and SQLJ). Discussion topics include writing stored procedures, writing user-defined functions, creating user-defined types, using triggers, and developing applications in partitioned environments or with federated systems.	SC09-2949 db2a0x70	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.	SC09-2950 db2l0x70	db2l0
Command Reference	Explains how to use the Command Line Processor and describes the DB2 commands that you can use to manage your database.	SC09-2951 db2n0x70	db2n0
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. This book also details how to use DRDA application servers with DB2 Connect application requesters. Available in HTML and PDF only.	No form number db2h1x70	db2h1

Table 35. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Data Movement Utilities Guide and Reference	Explains how to use DB2 utilities, such as import, export, load, AutoLoader, and DPROP, that facilitate the movement of data.	SC09-2955 db2dmx70	db2dm
Data Warehouse Center Administration Guide	Provides information on how to build and maintain a data warehouse using the Data Warehouse Center.	SC26-9993 db2ddx70	db2dd
Data Warehouse Center Application Integration Guide	Provides information to help programmers integrate applications with the Data Warehouse Center and with the Information Catalog Manager.	SC26-9994 db2adx70	db2ad
DB2 Connect User's Guide	Provides concepts, programming, and general usage information for the DB2 Connect products.	SC09-2954 db2c0x70	db2c0
DB2 Query Patroller Administration Guide	Provides an operational overview of the DB2 Query Patroller system, specific operational and administrative information, and task information for the administrative graphical user interface utilities.	SC09-2958 db2dwx70	db2dw
DB2 Query Patroller User's Guide	Describes how to use the tools and functions of the DB2 Query Patroller.	SC09-2960 db2wwx70	db2ww
Glossary	Provides definitions for terms used in DB2 and its components. Available in HTML format and in the *SQL Reference*.	No form number db2t0x70	db2t0
Image, Audio, and Video Extenders Administration and Programming	Provides general information about DB2 extenders, and information on the administration and configuration of the image, audio, and video (IAV) extenders and on programming using the IAV extenders. It includes reference information, diagnostic information (with messages), and samples.	SC26-9929 dmbu7x70	dmbu7
Information Catalog Manager Administration Guide	Provides guidance on managing information catalogs.	SC26-9995 db2dix70	db2di

Table 35. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Information Catalog Manager Programming Guide and Reference	Provides definitions for the architected interfaces for the Information Catalog Manager.	SC26-9997 db2bix70	db2bi
Information Catalog Manager User's Guide	Provides information on using the Information Catalog Manager user interface.	SC26-9996 db2aix70	db2ai
Installation and Configuration Supplement	Guides you through the planning, installation, and setup of platform-specific DB2 clients. This supplement also contains information on binding, setting up client and server communications, DB2 GUI tools, DRDA AS, distributed installation, the configuration of distributed requests, and accessing heterogeneous data sources.	GC09-2957 db2iyx70	db2iy
Message Reference	Lists messages and codes issued by DB2, the Information Catalog Manager, and the Data Warehouse Center, and describes the actions you should take. You can order both volumes of the Message Reference in the English language in North America with the form number SBOF-8932.	Volume 1 SC09-2978 db2m1x70 Volume 2 SC09-2979 db2m2x70	db2m0
OLAP Integration Server Administration Guide	Explains how to use the Administration Manager component of the OLAP Integration Server.	SC27-0782 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	Explains how to create and populate OLAP metaoutlines using the standard OLAP Metaoutline interface (not by using the Metaoutline Assistant).	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	Explains how to create OLAP models using the standard OLAP Model Interface (not by using the Model Assistant).	SC27-0783 db2lpx70	n/a
OLAP Setup and User's Guide	Provides configuration and setup information for the OLAP Starter Kit.	SC27-0702 db2ipx70	db2ip
OLAP Spreadsheet Add-in User's Guide for Excel	Describes how to use the Excel spreadsheet program to analyze OLAP data.	SC27-0786 db2epx70	db2ep

Table 35. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	Describes how to use the Lotus 1-2-3 spreadsheet program to analyze OLAP data.	SC27-0785 db2tpx70	db2tp
Replication Guide and Reference	Provides planning, configuration, administration, and usage information for the IBM Replication tools supplied with DB2.	SC26-9920 db2e0x70	db2e0
Spatial Extender User's Guide and Reference	Provides information about installing, configuring, administering, programming, and troubleshooting the Spatial Extender. Also provides significant descriptions of spatial data concepts and provides reference information (messages and SQL) specific to the Spatial Extender.	SC27-0701 db2sbx70	db2sb
SQL Getting Started	Introduces SQL concepts and provides examples for many constructs and tasks.	SC09-2973 db2y0x70	db2y0
SQL Reference, Volume 1 and Volume 2	Describes SQL syntax, semantics, and the rules of the language. This book 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-8933.	Volume 1 SC09-2974 db2s1x70 Volume 2 SC09-2975 db2s2x70	db2s0
System Monitor Guide and Reference	Describes how to collect different kinds of information about databases and the database manager. This book explains how to use the information to understand database activity, improve performance, and determine the cause of problems.	SC09-2956 db2f0x70	db2f0
Text Extender Administration and Programming	Provides general information about DB2 extenders and information on the administration and configuring of the text extender and on programming using the text extenders. It includes reference information, diagnostic information (with messages) and samples.	SC26-9930 desu9x70	desu9

Table 35. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	GC09-2850 db2p0x70	db2p0
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 7.	SC09-2976 db2q0x70	db2q0
	DB2 Installation and Configuration Information		
DB2 Connect Enterprise Edition for OS/2 and Windows Quick Beginnings	Provides planning, migration, installation, and configuration information for DB2 Connect Enterprise Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2953 db2c6x70	db2c6
DB2 Connect Enterprise Edition for UNIX Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Enterprise Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2952 db2cyx70	db2cy
DB2 Connect Personal Edition Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Personal Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for all supported clients.	GC09-2967 db2c1x70	db2c1
DB2 Connect Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Connect Personal Edition on all supported Linux distributions.	GC09-2962 db2c4x70	db2c4
DB2 Data Links Manager Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for AIX and Windows 32-bit operating systems.	GC09-2966 db2z6x70	db2z6

Table 35. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Enterprise - Extended Edition for UNIX Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2964 db2v3x70	db2v3
DB2 Enterprise - Extended Edition for Windows Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2963 db2v6x70	db2v6
DB2 for OS/2 Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the OS/2 operating system. This book also contains installation and setup information for many supported clients.	GC09-2968 db2i2x70	db2i2
DB2 for UNIX Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2970 db2ixx70	db2ix
DB2 for Windows Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2971 db2i6x70	db2i6
DB2 Personal Edition Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on the OS/2 and Windows 32-bit operating systems.	GC09-2969 db2i1x70	db2i1
DB2 Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on all supported Linux distributions.	GC09-2972 db2i4x70	db2i4

Table 35. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Query Patroller Installation Guide	Provides installation information about DB2 Query Patroller.	GC09-2959 db2iwx70	db2iw
DB2 Warehouse Manager Installation Guide	Provides installation information for warehouse agents, warehouse transformers, and the Information Catalog Manager.	GC26-9998 db2idx70	db2id
	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. The sample programs are provided for informational purposes only. Not all samples are available in all programming languages. The HTML samples are only available when the DB2 Application Development Client is installed. For more information on the programs, refer to the *Application Building Guide*.	No form number	db2hs
	Release Notes		
DB2 Connect Release Notes	Provides late-breaking information that could not be included in the DB2 Connect books.	See note #2.	db2cr
DB2 Installation Notes	Provides late-breaking installation-specific information that could not be included in the DB2 books.	Available on product CD-ROM only.	
DB2 Release Notes	Provides late-breaking information about all DB2 products and features that could not be included in the DB2 books.	See note #2.	db2ir

Notes:

1. The character *x* in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d0e70` identifies the English version of the *Administration Guide* and the file name `db2d0f70` identifies the French version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

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. Late breaking information that could not be included in the DB2 books is available in the Release Notes in HTML format and as an ASCII file. The HTML version is available from the Information Center and on the product CD-ROMs. To view the ASCII file:
   - On UNIX-based platforms, see the `Release.Notes` file. This file is located in the `DB2DIR/Readme/%L` directory, where `%L` represents the locale name and `DB2DIR` represents:
     – `/usr/lpp/db2_07_01` on AIX
     – `/opt/IBMdb2/V7.1` on HP-UX, PTX, Solaris, and Silicon Graphics IRIX
     – `/usr/IBMdb2/V7.1` on Linux.
   - On other platforms, see the `RELEASE.TXT` file. This file is located in the directory where the product is installed. On OS/2 platforms, you can also double-click the **IBM DB2** folder and then double-click the **Release Notes** icon.

Printing the PDF Books

If you prefer to have printed copies of the books, you can print the PDF files found on the DB2 publications CD-ROM. Using the Adobe Acrobat Reader, you can print either the entire book or a specific range of pages. For the file name of each book in the library, see Table 35 on page 268.

You can obtain the latest version of the Adobe Acrobat Reader from the Adobe Web site at http://www.adobe.com.

The PDF files are included on the DB2 publications CD-ROM with a file extension of PDF. To access the PDF files:

1. Insert the DB2 publications CD-ROM. On UNIX-based platforms, mount the DB2 publications CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
2. Start the Acrobat Reader.
3. Open the desired PDF file from one of the following locations:
   - On OS/2 and Windows platforms:

     `x:\doc\language` directory, where *x* represents the CD-ROM drive and *language* represent the two-character country code that represents your language (for example, EN for English).
   - On UNIX-based platforms:

     `/cdrom/doc/%L` directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

You can also copy the PDF files from the CD-ROM to a local or network drive and read them from there.

Ordering the Printed Books

You can order the printed DB2 books either individually or as a set (in North America only) by using a sold bill of forms (SBOF) number. To order books, 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. You can also order the books from the Publications Web page at http://www.elink.ibmlink.ibm.com/pbl/pbl.

Two sets of books are available. SBOF-8935 provides reference and usage information for the DB2 Warehouse Manager. SBOF-8931 provides reference and usage information for all other DB2 Universal Database products and features. The contents of each SBOF are listed in the following table:

Table 36. Ordering the printed books

SBOF Number	Books Included	
SBOF-8931	• Administration Guide: Planning • Administration Guide: Implementation • Administration Guide: Performance • Administrative API Reference • Application Building Guide • Application Development Guide • CLI Guide and Reference • Command Reference • Data Movement Utilities Guide and Reference • Data Warehouse Center Administration Guide • Data Warehouse Center Application Integration Guide • DB2 Connect User's Guide • Installation and Configuration Supplement • Image, Audio, and Video Extenders Administration and Programming • Message Reference, Volumes 1 and 2	• OLAP Integration Server Administration Guide • OLAP Integration Server Metaoutline User's Guide • OLAP Integration Server Model User's Guide • OLAP Integration Server User's Guide • OLAP Setup and User's Guide • OLAP Spreadsheet Add-in User's Guide for Excel • OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3 • Replication Guide and Reference • Spatial Extender Administration and Programming Guide • SQL Getting Started • SQL Reference, Volumes 1 and 2 • System Monitor Guide and Reference • Text Extender Administration and Programming • Troubleshooting Guide • What's New
SBOF-8935	• Information Catalog Manager Administration Guide • Information Catalog Manager User's Guide • Information Catalog Manager Programming Guide and Reference	• Query Patroller Administration Guide • Query Patroller User's Guide

DB2 Online Documentation

Accessing Online Help

Online help is available with all DB2 components. The following table describes the various types of help.

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* represents 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.
Client Configuration Assistant Help *Command Center Help* *Control Center Help* *Data Warehouse Center Help* *Event Analyzer Help* *Information Catalog Manager Help* *Satellite Administration Center Help* *Script Center Help*	Explains the tasks you can perform in a window or notebook. The help includes overview and prerequisite information you need to know, and it 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* represents 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* represents the file where you want to save the message help.

Type of Help	Contents	How to Access...
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: `help` *`statement`* where *statement* represents 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* represents a valid five-digit SQL state and *class code* represents 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.

Viewing Information Online

The books 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:

- If you are running DB2 administration tools, use the Information Center.
- From a browser, click **File —>Open Page**. The page you open contains descriptions of and links to DB2 information:
  - On UNIX-based platforms, open the following page:

    ```
    INSTHOME/sqllib/doc/%L/html/index.htm
    ```

    where `%L` represents 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 the **DB2 Information** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

Installing the Netscape Browser

If you do not already have a Web browser installed, you can install Netscape from the Netscape CD-ROM found in the product boxes. For detailed instructions on how to install it, perform the following:

1. Insert the Netscape CD-ROM.
2. On UNIX-based platforms only, mount the CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
3. For installation instructions, refer to the `CDNAV`*nn*`.txt` file, where *nn* represents your two character language identifier. The file is located at the root directory of the CD-ROM.

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.

You can open the Information Center by double-clicking the Information Center icon. Depending on the system you are using, the icon is in the Information folder in the main product folder or the Windows **Start** menu.

You can also access the Information Center by using the toolbar and the **Help** menu on the DB2 Windows platform.

The Information Center provides six types of information. Click the appropriate tab to look at the topics provided for that type.

Tasks	Key tasks you can perform using DB2.
Reference	DB2 reference information, such as keywords, commands, and APIs.
Books	DB2 books.
Troubleshooting	Categories of error messages and their recovery actions.
Sample Programs	Sample programs that come with the DB2 Application Development Client. If you did not install the DB2 Application Development Client, this tab is not displayed.
Web	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 a find feature, so you can look for a specific topic without browsing the lists.

For a full text search, follow the hypertext link in the Information Center to the **Search DB2 Online Information** search form.

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 using one of the following methods:

On Windows
Click **Start** and select **Programs —> IBM DB2 —> Information —> Start HTML Search Server**.

On OS/2
Double-click the **DB2 for OS/2** folder, and then double-click the **Start HTML Search Server** icon.

Refer to the release notes if you experience any other problems when searching the HTML information.

Note: The Search function is not available in the Linux, PTX, and Silicon Graphics IRIX environments.

Using DB2 Wizards

Wizards help you complete specific administration tasks by taking you through each task one step at a time. Wizards are available through the Control Center and the Client Configuration Assistant. The following table lists the wizards and describes their purpose.

Note: The Create Database, Create Index, Configure Multisite Update, and Performance Configuration wizards are available for the partitioned database environment.

Wizard	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, right-click the database you want to back up and select **Backup —> Database Using Wizard**.

Wizard	Helps You to...	How to Access...
Configure Multisite Update	Configure a multisite update, a distributed transaction, or a two-phase commit.	From the Control Center, right-click the **Databases** folder and select **Multisite Update**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, right-click the **Databases** folder and select **Create —> Database Using Wizard**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, right-click the **Tables** icon and select **Create —> Table Using Wizard**.
Create Table Space	Create a new table space.	From the Control Center, right-click the **Table Spaces** icon and select **Create —> Table Space Using Wizard**.
Create Index	Advise which indexes to create and drop for all your queries.	From the Control Center, right-click the **Index** icon and select **Create —> Index Using Wizard**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, right-click the database you want to tune and select **Configure Performance Using Wizard**. For the partitioned database environment, from the Database Partitions view, right-click the first database partition you want to tune and select **Configure Performance Using Wizard**.
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, right-click the database you want to restore and select **Restore —> Database Using Wizard**.

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, perform the following steps:

1. Copy all files and subdirectories from `\sqllib\doc\html` on your local system to a Web server. Each book has its own subdirectory that contains 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, refer to the NetQuestion Appendix in the *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. When you are able to view the book files, you can bookmark commonly viewed topics. 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 how you can serve the DB2 Universal Database online documentation files from a central machine, refer to the NetQuestion Appendix in the *Installation and Configuration Supplement*.

Searching Information Online

To find information in the HTML files, use one of the following methods:

- Click **Search** in the top frame. Use the search form to find a specific topic. This function is not available in the Linux, PTX, or Silicon Graphics IRIX environments.
- Click **Index** in the top frame. Use the index to find a specific topic in the book.
- Display the table of contents or index of the help or 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 281 for details.

Appendix F. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

Trademarks

Index

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. This guide suggests information that you can gather to help DB2 Customer Support 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.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 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.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

ftp.software.ibm.com
Log on as anonymous. In the directory `/ps/products/db2`, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l
These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2
Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the *IBM Software Support Handbook*. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

SC09-2946-01

Spine information:

IBM® DB2® Universal Database Administration Guide: Planning Version 7