IBM® DB2 Universal Database™

Administration Guide: Planning

Version 8

SC09-4822-00

IBM® DB2 Universal Database™

Administration Guide: Planning

Version 8

SC09-4822-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

Contents

About this book

The Administration Guide in its three volumes provides information necessary to use and administer the 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 use a graphical user interface to 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, which allows you 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, select the Control Center icon from the DB2 folder, or use the Start menu 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 Health Center provides a tool to assist DBAs in the resolution of performance and resource allocation problems.
- The Tools 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:

Database Concepts

- Chapter 1, “Basic relational database concepts”, presents an overview of database objects and database concepts.
- Chapter 2, “Parallel database systems”, provides an introduction to the types of parallelism available with DB2.
- Chapter 3, “About data warehousing”, provides an overview of data warehousing and data warehousing tasks.

Database Design

- Chapter 4, “Logical database design”, discusses the concepts and guidelines for logical database design.
- Chapter 5, “Physical database design”, discusses the guidelines for physical database design, including space requirements and table space design.
- Chapter 6, “Designing Distributed Databases”, discusses how you can access multiple databases in a single transaction.
- Chapter 7, “Designing for XA-compliant transaction managers”, discusses how you can use your databases in a distributed transaction processing environment.

Appendixes

- Appendix A, “Incompatibilities between releases”, presents the incompatibilities introduced by Version 7 and Version 8, as well as future incompatibilities that you should be aware of.
- Appendix B, “National language support (NLS)”, introduces DB2 National Language Support, including information about territories, languages, and code pages.

A brief overview of the other Administration Guide volumes

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:

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.

Appendixes

- ″Naming Rules″ presents the rules to follow when naming databases and objects.
- ″Lightweight Directory Access Protocol (LDAP) Directory Services″ provides information about how you can use LDAP Directory Services.
- ″Issuing Commands to Multiple Database Partition″ discusses the use of the *db2_all* and *rah* shell scripts to send commands to all partitions in a partitioned database environment.
- ″Windows Management Instrumentation (WMI) Support″ describes how DB2 supports this management infrastructure standard to integrate various hardware and software management systems. Also discussed is how DB2 integrates with WMI.
- ″How DB2 for Windows NT Works with Windows NT Security″ describes how DB2 works with Windows NT security.

- ″Using the Windows Performance Monitor″ provides information about registering DB2 with the Windows NT Performance Monitor, and using the performance information.
- ″Working with Windows 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.
- ″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.

Note: Two chapters have been removed from this book.

All of the information on the DB2 utilities for moving data, and the comparable topics from the *Command Reference* and the *Administrative API Reference*, have been consolidated into the *Data Movement Utilities Guide and Reference*.

The *Data Movement Utilities Guide and Reference* is your primary, single source of information for these topics.

To find out more about replication of data, see *Replication Guide and Reference*.

All of the information on the methods and tools for backing up and recovering data, and the comparable topics from the *Command Reference* and the *Administrative API Reference*, have been consolidated into the *Data Recovery and High Availability Guide and Reference*.

The *Data Recovery and High Availability Guide and Reference* is your primary, single source of information for these topics.

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

- ″Introduction to Performance″ introduces concepts and considerations for managing and improving DB2 UDB performance.
- ″Architecture and Processes″ 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 database manager, database, and DAS 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. Database concepts

Chapter 1. Basic relational database concepts

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. 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.

Database partition groups:

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

In earlier versions of DB2, *database partition groups* were known as *nodegroups*.

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. 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), 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

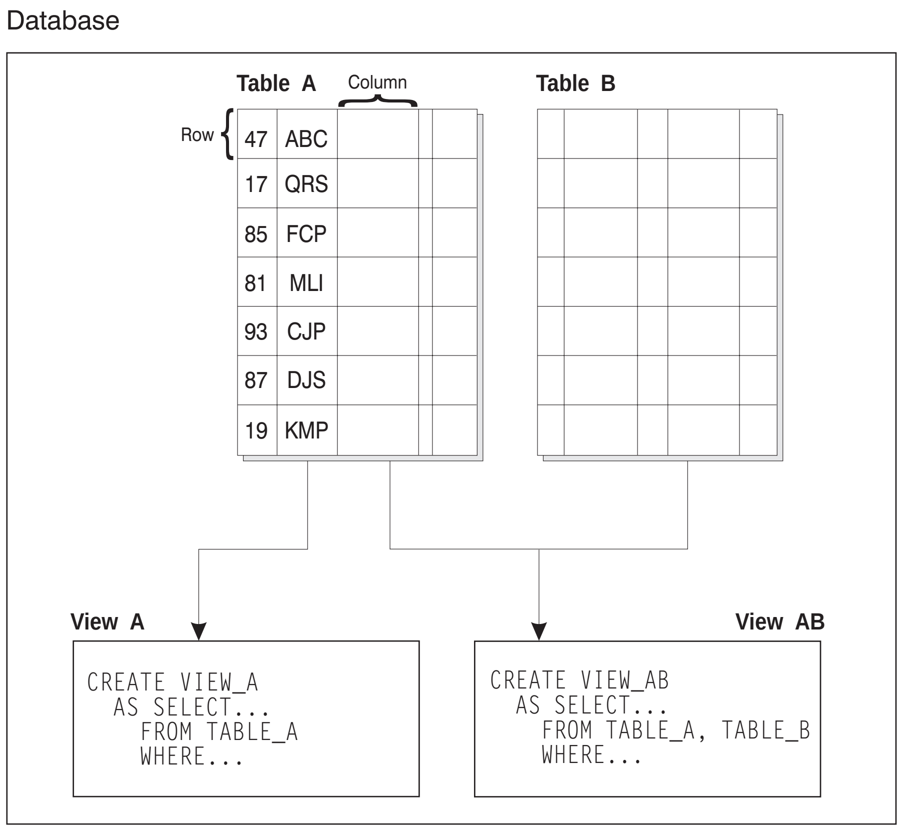
```
SELECT <data_name> FROM <table_name>
```

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 1 shows the relationship between tables and views.



```
CREATE VIEW_A
  AS SELECT...
    FROM TABLE_A
    WHERE...
```

```
CREATE VIEW_AB
  AS SELECT...
    FROM TABLE_A, TABLE_B
    WHERE...
```

Figure 1. 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 2 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.

Figure 2 shows the relationship between an index and a table.

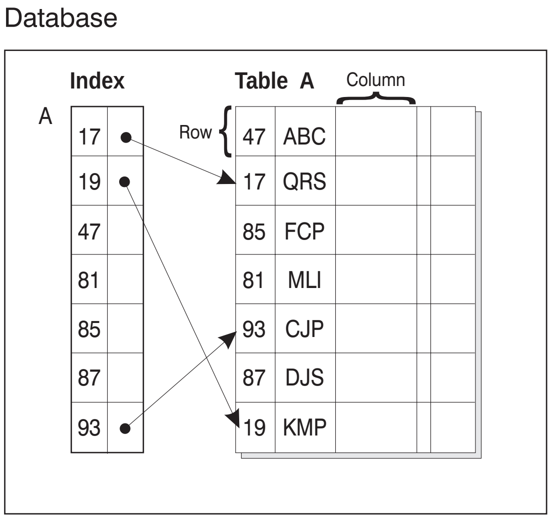


Figure 2. Relationship Between an Index and a Table

Figure 3 on page 6 illustrates the relationships among some database objects. It also shows that tables, indexes, and long data are stored in table spaces.

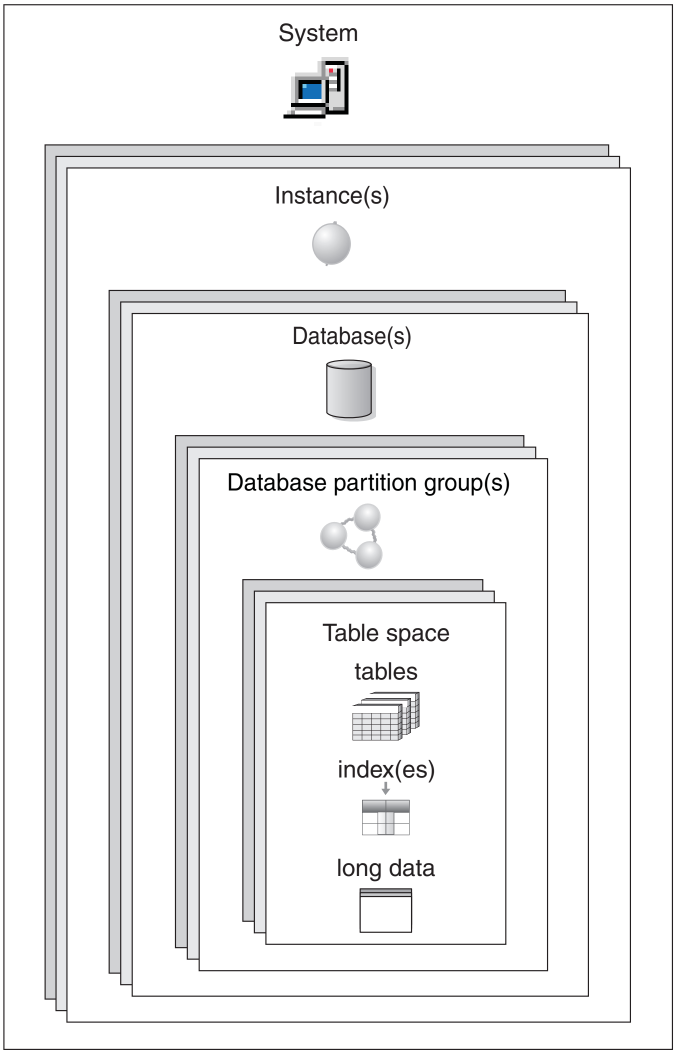


Figure 3. Relationships Among Some Database Objects

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.

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:

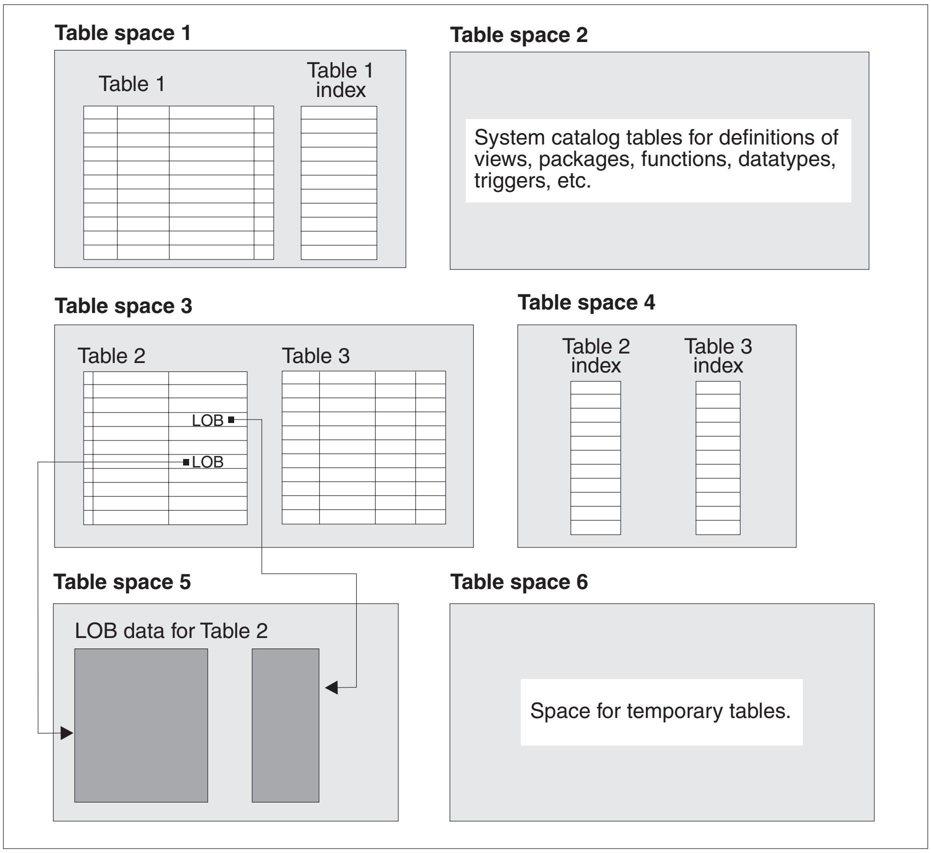


Figure 4. Table Spaces

Table spaces reside in database partition groups. Table space definitions and attributes are recorded in the database system catalog.

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.

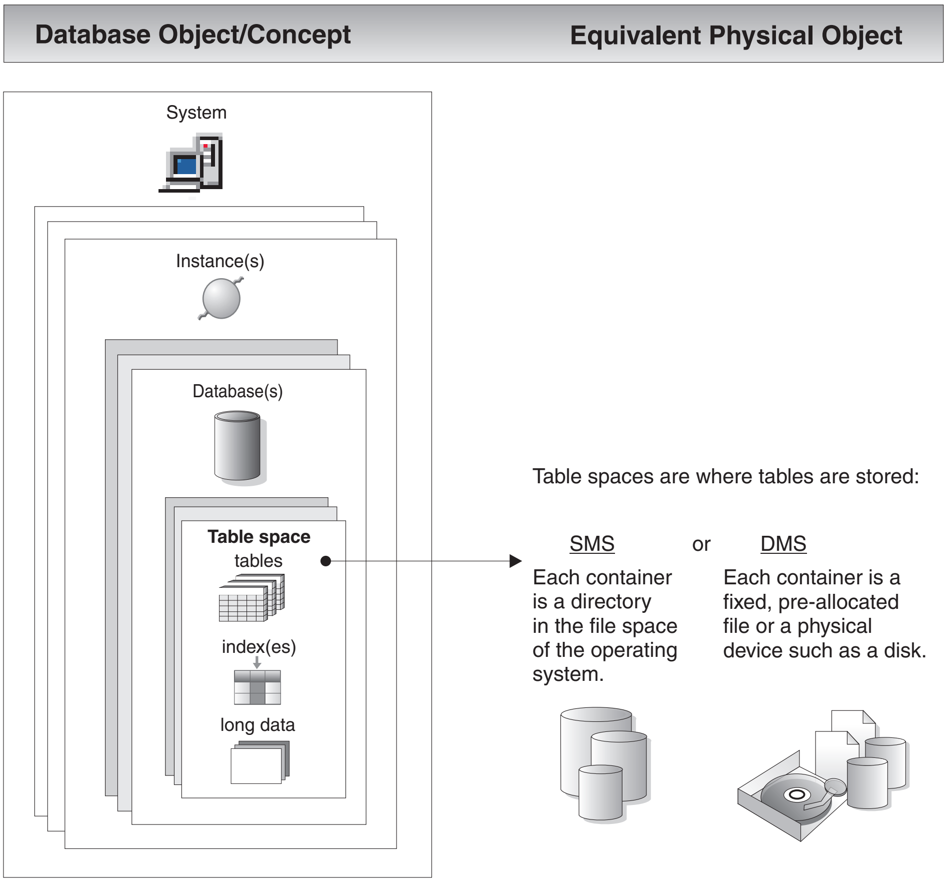

Database Object/Concept
Equivalent Physical Object
System
Instance(s)
Database(s)
Table space
tables
index(es)
long data
Table spaces are where tables are stored:
SMS
or
DMS
Each container is a directory in the file space of the operating system.
Each container is a fixed, pre-allocated file or a physical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

Figure 5. Table Spaces and Tables

Figure 6 on page 10 shows the three table space types: *regular*, *temporary*, and *large*.

Tables containing user data exist in regular table spaces. The default user table space is called USERSPACE1. 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 large object data, such as multimedia objects, exist in large table spaces or in regular table spaces. The base column data for these columns is stored in a regular table space, while the long field or large object data can be stored in the same regular table space or in a specified large table space.

Indexes can be stored in regular table spaces or large 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.

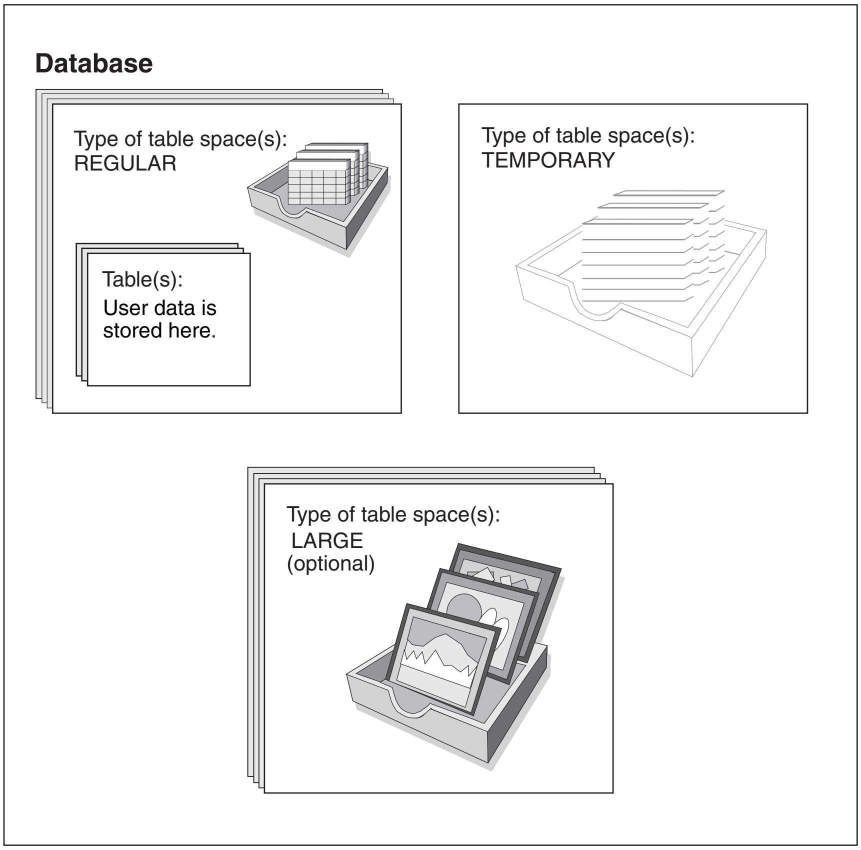

Database
Type of table space(s): REGULAR
Table(s): User data is stored here.
Type of table space(s): TEMPORARY
Type of table space(s): LARGE (optional)

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.

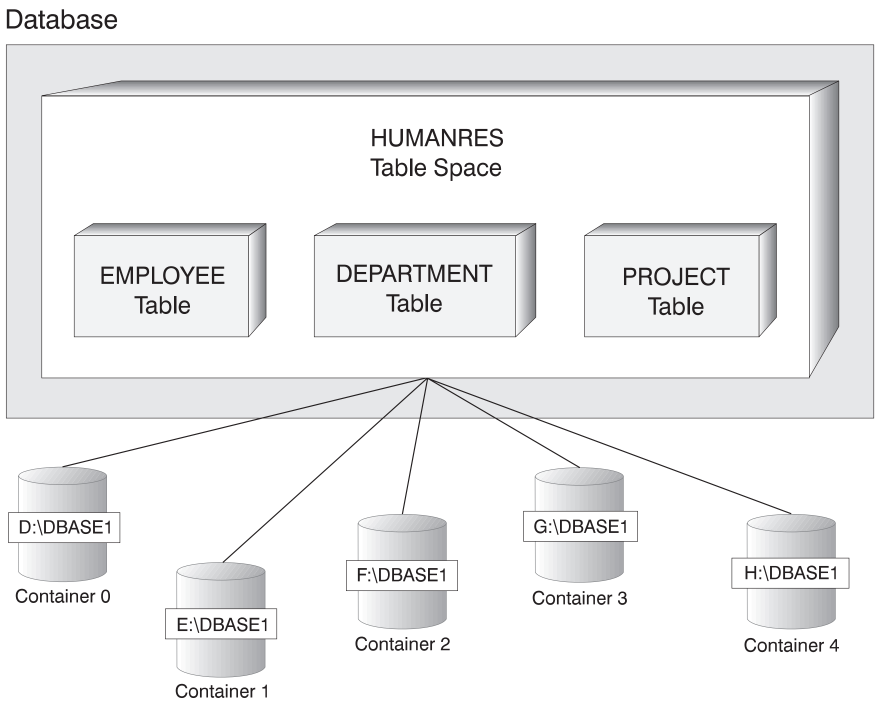


Figure 7. Table Spaces and Tables Within a Database

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 pools:

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.

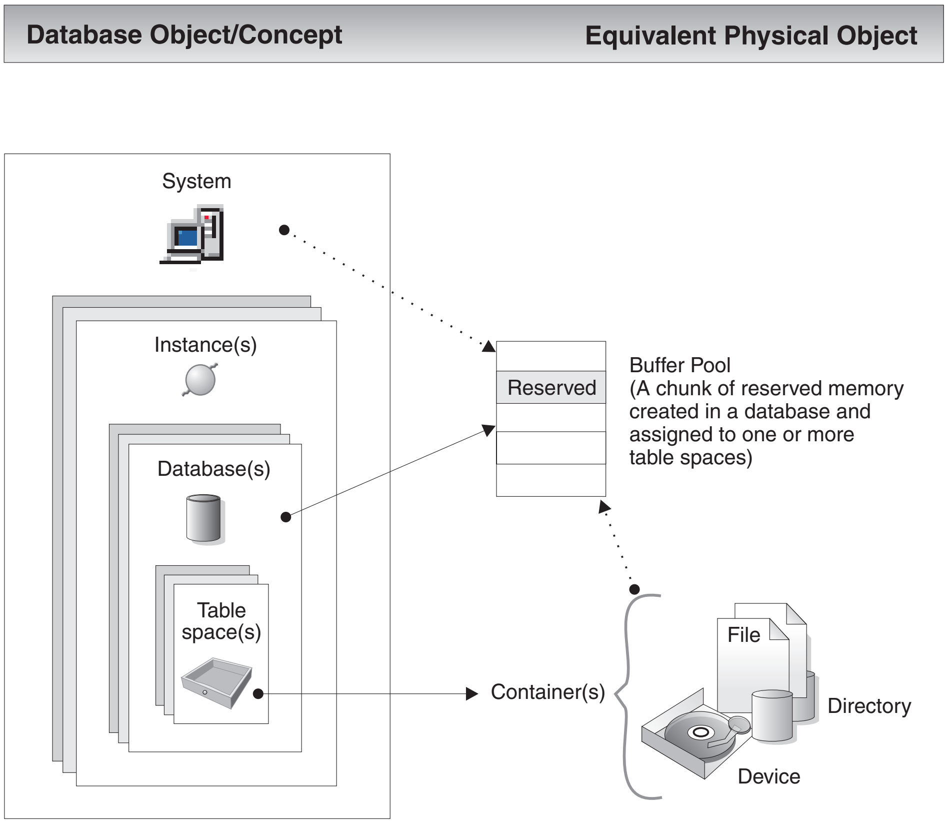

Database Object/Concept
Equivalent Physical Object
System
Instance(s)
Database(s)
Table space(s)
Reserved
Buffer Pool (A chunk of reserved memory created in a database and assigned to one or more table spaces)
Container(s)
File
Directory
Device

Figure 8. Buffer Pool and Containers

Related concepts:

- "Indexes" in the *SQL Reference, Volume 1*
- "Tables" in the *SQL Reference, Volume 1*
- "Relational databases" in the *SQL Reference, Volume 1*
- "Schemas" in the *SQL Reference, Volume 1*
- "Views" in the *SQL Reference, Volume 1*

- “Table spaces and other storage structures” in the *SQL Reference, Volume 1*

Configuration parameters

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 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
- the database configuration file for each individual database.

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.

Database manager configuration parameters are stored in a file named `db2systm`. This file is created when the instance of the database manager is created. In UNIX-based environments, this file can be found in the `sqllib` subdirectory for the instance of the database manager. In Windows, the default location of this file is the instance subdirectory of the `sqllib` directory. If the DB2INSTPROF variable is set, the file is in the `instance` subdirectory of the directory specified by the DB2INSTPROF variable.

In a partitioned database environment, this file resides on a shared file system so that all database partition servers have access to the same file. The configuration of the database manager is the same on all database partition servers.

Most of the parameters either affect the amount of system resources that will be allocated to a single instance of the database manager, or they configure the setup of the database manager and the different communications subsystems based on environmental considerations. In addition, there are other parameters that serve informative purposes only and cannot be changed. All of these parameters have global applicability independent of any single database stored under that instance of the database manager.

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.

Parameters for an individual database are stored in a configuration file named `SQLDBCON`. This file is stored along with other control files for the database in the `SQLnnnnn` directory, where `nnnnn` is a number assigned when the database was created. Each database has its own configuration file, and most of the parameters in the file specify the amount of resources allocated to that database. The file also contains descriptive information, as well as flags that indicate the status of the database.

In a partitioned database environment, a separate `SQLDBCON` file exists for each database partition. The values in the `SQLDBCON` file may be the same or different at each database partition, but the recommendation is that the database configuration parameter values be the same on all partitions.

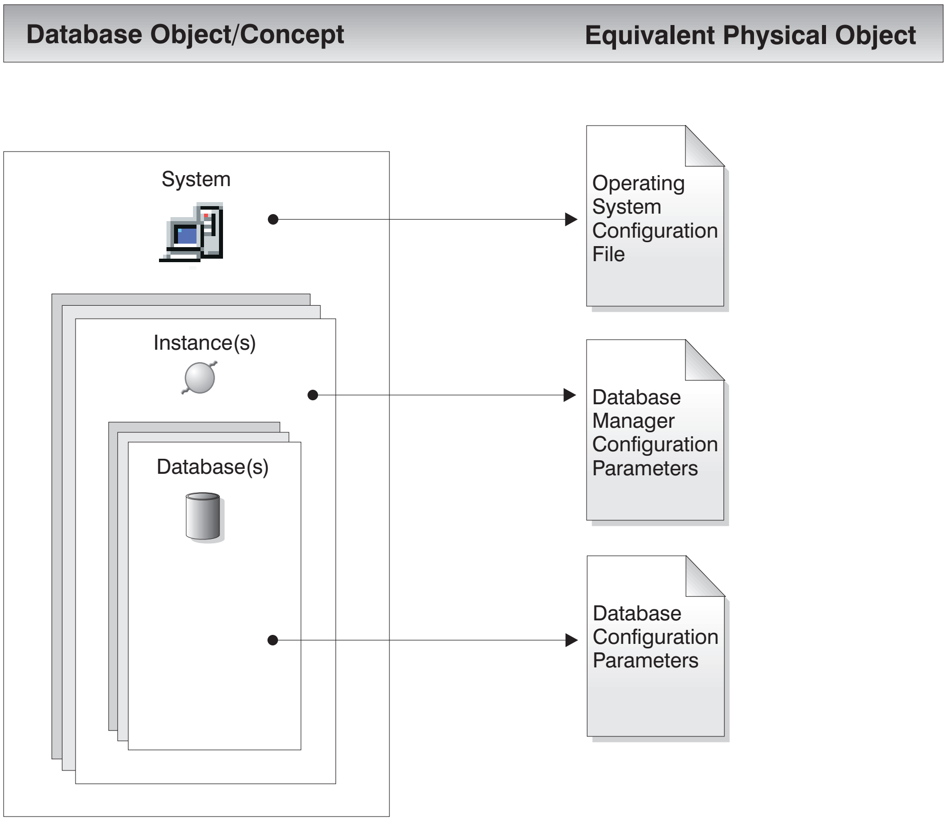

Database Object/Concept
Equivalent Physical Object
System
Instance(s)
Database(s)
Operating System Configuration File
Database Manager Configuration Parameters
Database Configuration Parameters

Figure 9. Configuration Parameter Files

Related concepts:

- "Configuration parameter tuning" in the *Administration Guide: Performance*

Related tasks:

- "Configuring DB2 with configuration parameters" in the *Administration Guide: Performance*

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.

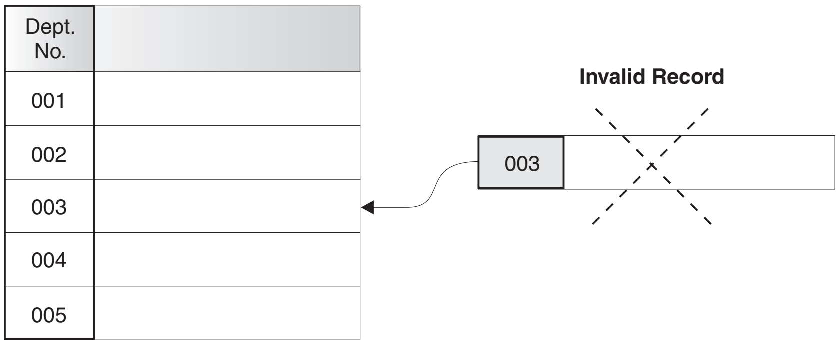


Figure 10. Unique Constraints Prevent Duplicate Data

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
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.

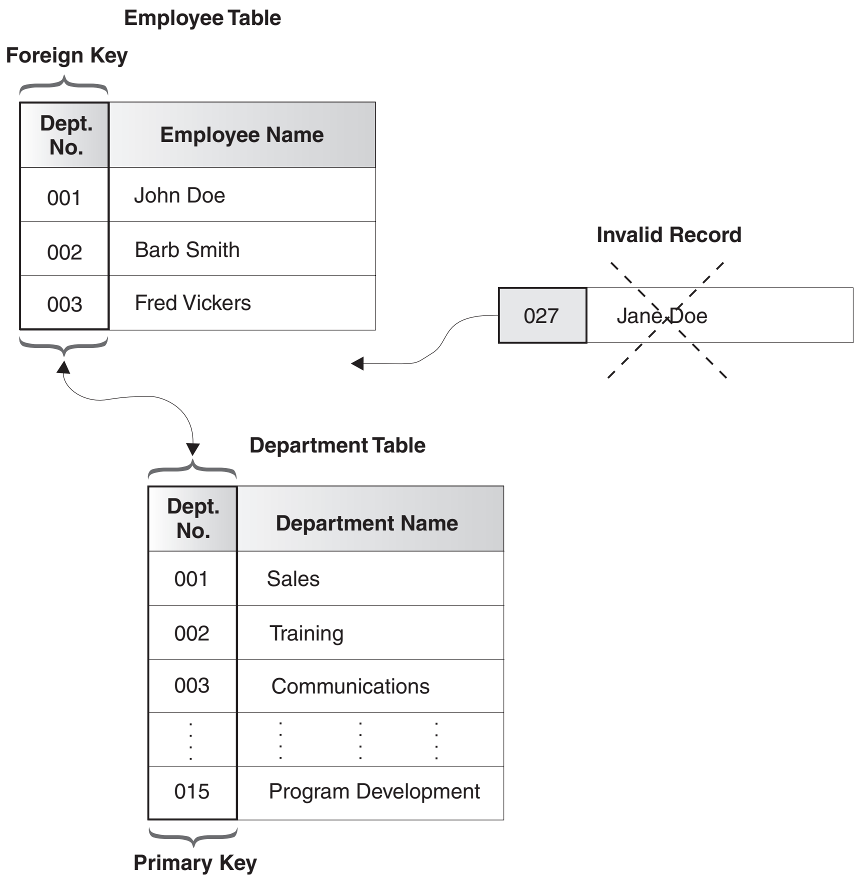


Figure 11. Foreign and Primary Key Constraints Define Relationships and Protect Data

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.

Related concepts:

- "Constraints" on page 80
- "Triggers" on page 85

Developing a Backup and Recovery Strategy

A database can become unusable because of hardware or software failure, or both. You may, at one time or another, encounter storage problems, power interruptions, and application failures, and different failure scenarios require different recovery actions. Protect your data against the possibility of loss by having a well rehearsed recovery strategy in place. Some of the questions that you should answer when developing your recovery strategy are: Will the database be recoverable? How much time can be spent recovering the database? How much time will pass between backup operations? How much storage space can be allocated for backup copies and archived logs? Will table space level backups be sufficient, or will full database backups be necessary?

A database recovery strategy should ensure that all information is available when it is required for database recovery. It should include a regular schedule for taking database backups and, in the case of partitioned database systems, include backups when the system is scaled (when database partition servers or nodes are added or dropped). Your overall strategy should also include procedures for recovering command scripts, applications, user-defined functions (UDFs), stored procedure code in operating system libraries, and load copies.

Different recovery methods are discussed in the sections that follow, and you will discover which recovery method is best suited to your business environment.

The concept of a database *backup* is the same as any other data backup: taking a copy of the data and then 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. You can then rebuild the database if it becomes damaged or corrupted in some way.

The rebuilding of the database is called *recovery*. *Version recovery* is the restoration of a previous version of the database, using an image that was created during a backup operation. *Rollforward recovery* is the reapplication of transactions recorded in the database log files after a database or a table space backup image has been restored.

Crash recovery is the automatic recovery of the database if a failure occurs before all of the changes that are part of one or more units of work (transactions) are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

Recovery log files and the recovery history file are created automatically when a database is created (Figure 12). These log files are important if you need to recover data that is lost or damaged. You cannot directly modify a recovery log file or the recovery history file; however, you can delete entries from the recovery history file using the PRUNE HISTORY command. You can also use the *rec_his_retentn* database configuration parameter to specify the number of days that the recovery history file will be retained.

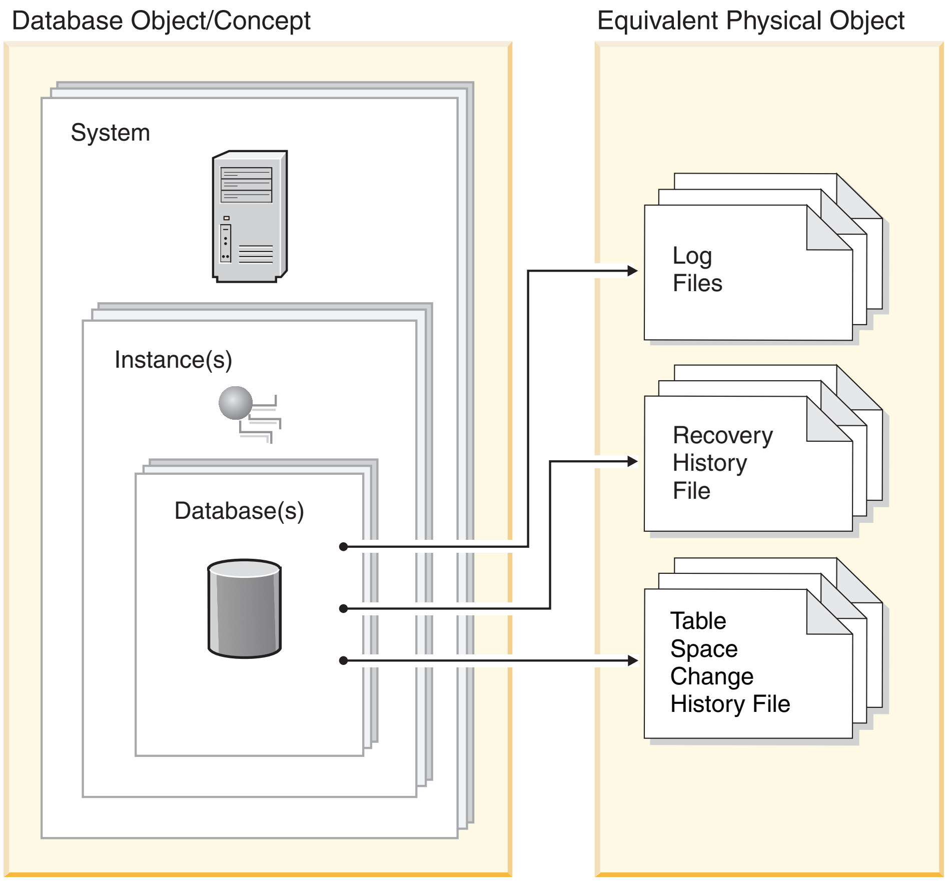


Figure 12. Recovery Log Files and the Recovery History File

Each database includes *recovery logs*, which are used to recover from application or system errors. In combination with the database backups, they are used to recover the consistency of the database right up to the point in time when the error occurred.

The *recovery history file* contains a summary of the backup information that can be used to determine recovery options, if 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 and restore operations, among others. This file is located in the database directory.

The *table space change history file*, which is also located in the database directory, contains information that can be used to determine which log files are required for the recovery of a particular table space.

Data that is easily recreated can be stored in a non-recoverable database. This includes data from an outside source that is used for read-only applications, and tables that are not often updated, for which the small amount of logging does not justify the added complexity of managing log files and rolling forward after a restore operation. *Non-recoverable databases* have both the *logretain* and the *userexit* database configuration parameters disabled. This means that the only logs that are kept are those required for crash recovery. These logs are known as *active logs*, and they contain current transaction data. Version recovery using *offline* backups is the primary means of recovery for a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken and rollforward recovery is not supported.

Data that *cannot* be easily recreated should be stored in a recoverable database. This includes data whose source is destroyed after the data is loaded, data that is manually entered into tables, and data that is modified by application programs or users after it is loaded into the database. *Recoverable databases* have either the *logretain* database configuration parameter set to "RECOVERY", the *userexit* database configuration parameter enabled, or both. Active logs are still available for crash recovery, but you also have the *archived logs*, which contain committed transaction data. Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken. However, with rollforward recovery, you can roll the database *forward* (that is, past the time when the backup image was taken) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

Recoverable database backup operations can be performed either offline or *online* (online meaning that other applications can connect to the database during the backup operation). Database restore and rollforward operations must always be performed offline. During an online backup operation, rollforward recovery ensures that *all* table changes are captured and reapplied if that backup is restored.

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or rollforward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

Related concepts:

- "Crash Recovery" in the *Data Recovery and High Availability Guide and Reference*
- "Version Recovery" in the *Data Recovery and High Availability Guide and Reference*
- "Rollforward Recovery" in the *Data Recovery and High Availability Guide and Reference*
- "Data Links server file backups" in the *Post V8 GA*
- "Failure and recovery overview" in the *DB2 Data Links Manager Administration Guide and Reference*

Related reference:

- "Recovery History Retention Period configuration parameter - rec_his_retentn" in the *Administration Guide: Performance*
- "DB2 Data Links Manager system setup and backup recommendations" in the *DB2 Data Links Manager Administration Guide and Reference*

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.

Related concepts:

- "Authentication" on page 23
- "Authorization" on page 24

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.

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.
- 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 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 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.

Related concepts:

- "Security" on page 22

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.

Related concepts:

- "Authorization and privileges" in the *SQL Reference, Volume 1*
- "Privileges, authorities, and authorization" in the *Administration Guide: Implementation*
- "Security" on page 22

Chapter 2. Parallel database systems

Data partitioning

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.

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 database partition groups, 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 database partition group 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.

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 database partition group 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.

An alternative to consider when you want tables to be positioned on each database partition, is to use materialized query tables and then replicate those tables. You can create a materialized query table containing the information that you need, and then replicate it to each node.

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. The following types of parallelism are supported by DB2:

- I/O
- Query
- Utility

Input/output 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.

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.

Intra-partition parallelism

Intra-partition parallelism refers to the ability to break up a query into multiple parts. Some DB2 utilities also perform this type of parallelism.

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 13 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.

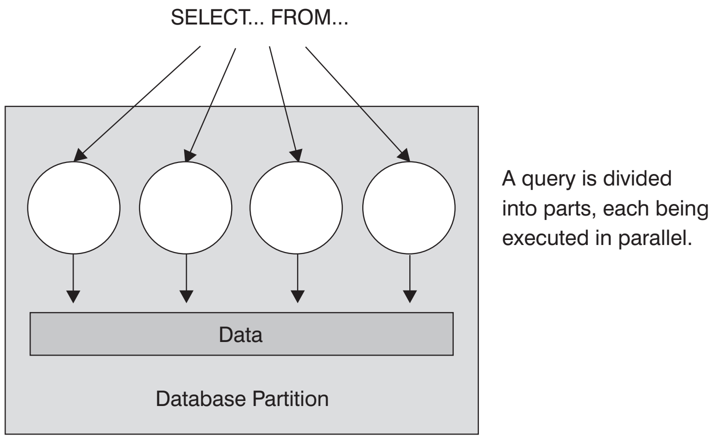

SELECT... FROM...
A query is divided into parts, each being executed in parallel.
Data
Database Partition

Figure 13. 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 DB2 utilities also perform this type of parallelism.

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 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 on a single partition.

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

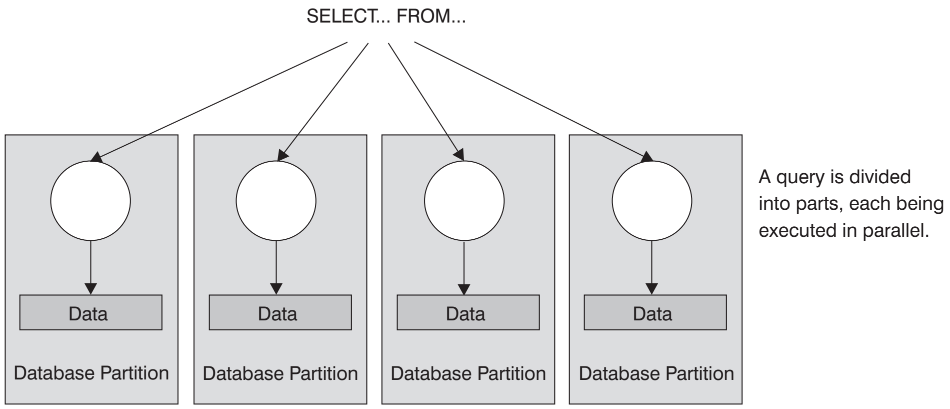


Figure 14. Inter-partition Parallelism

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:

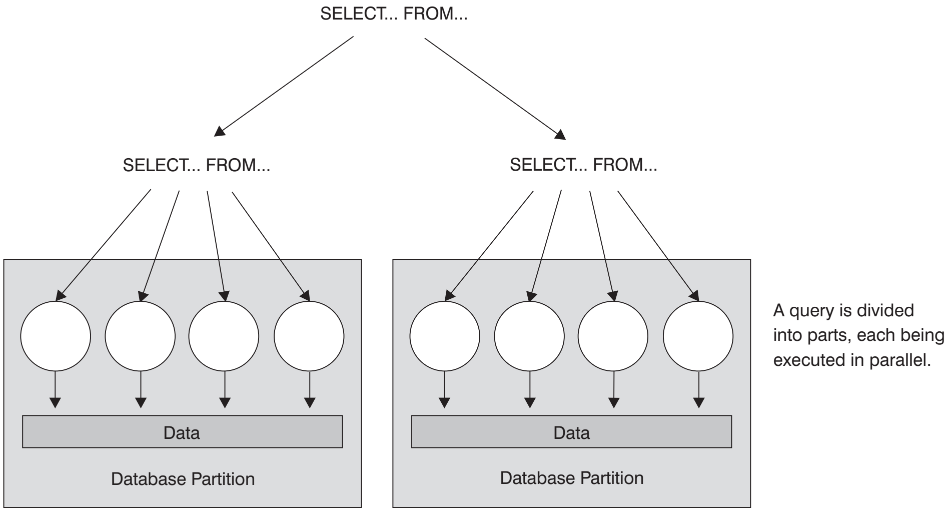


Figure 15. Simultaneous Inter-partition and Intra-partition Parallelism

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.

In a partitioned database environment, the LOAD command takes advantage of intra-partition, inter-partition, and I/O parallelism by parallel invocations at each database partition where the table resides.

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.

Related concepts:

- "Partition and processor environments" on page 30

Partition and processor 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 16 on page 31). 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.

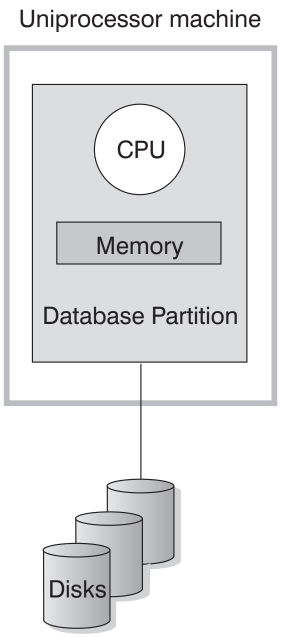


Figure 16. Single Partition On a Single Processor

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 17 on page 32), 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.

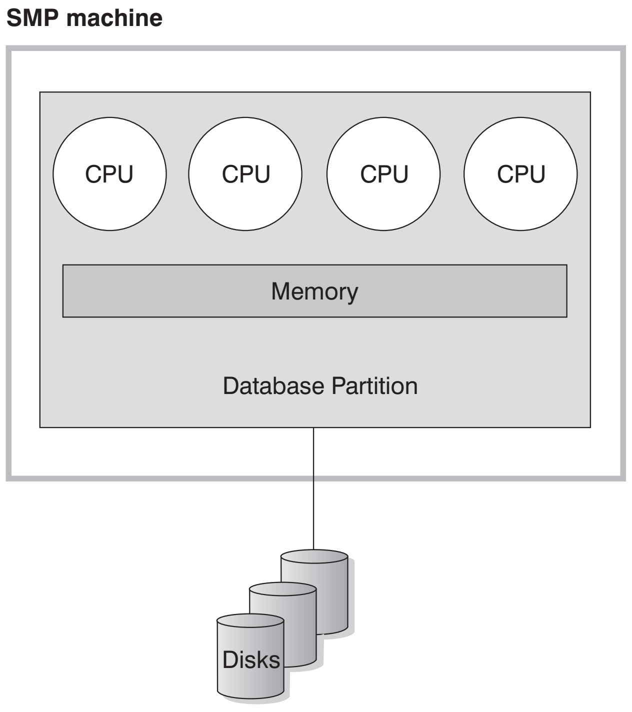


Figure 17. Single Partition Database Symmetric Multiprocessor System

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 18 on page 34). 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.

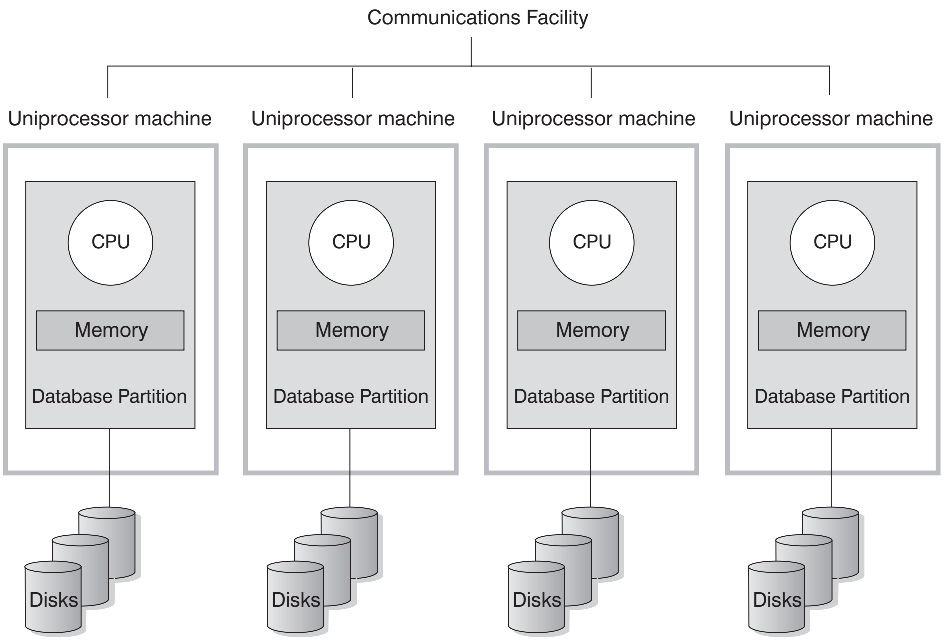


Figure 18. Massively Parallel Processing System

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 19 on page 35).

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.

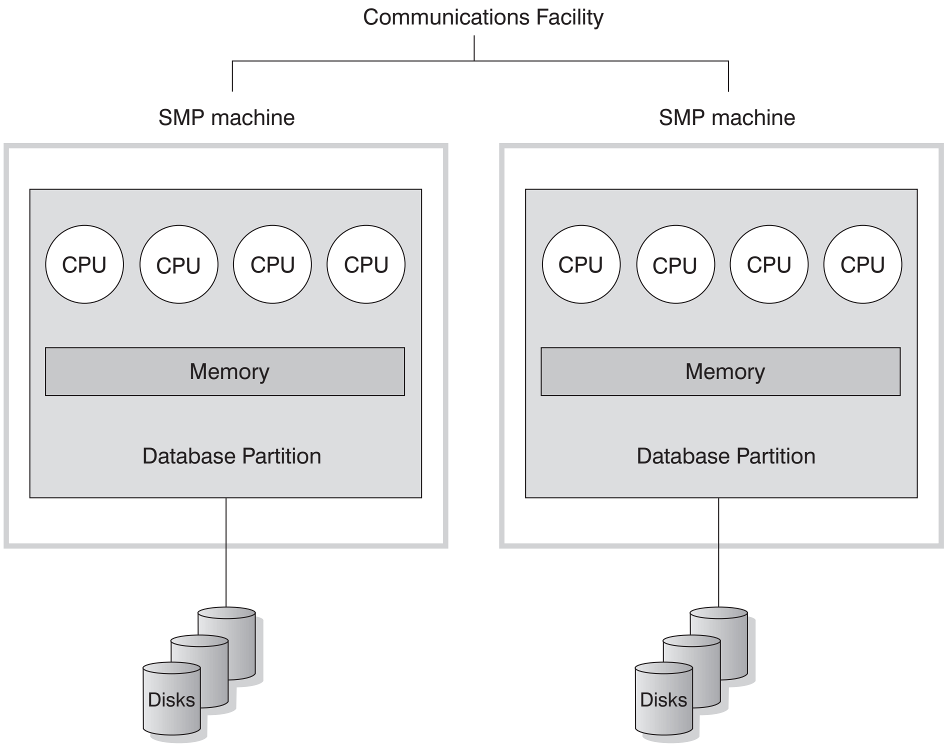


Figure 19. Cluster of SMPs

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 20 on page 36 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.

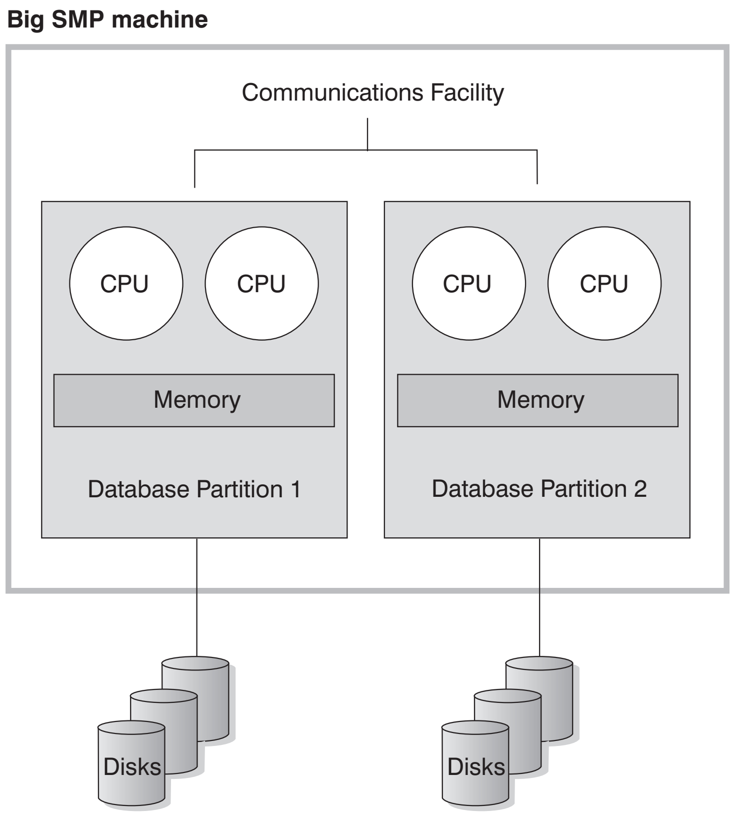


Figure 20. Partitioned Database, Symmetric Multiprocessor System

Figure 21 on page 37 illustrates the fact that you can multiply the configuration shown in Figure 20 to increase processing power.

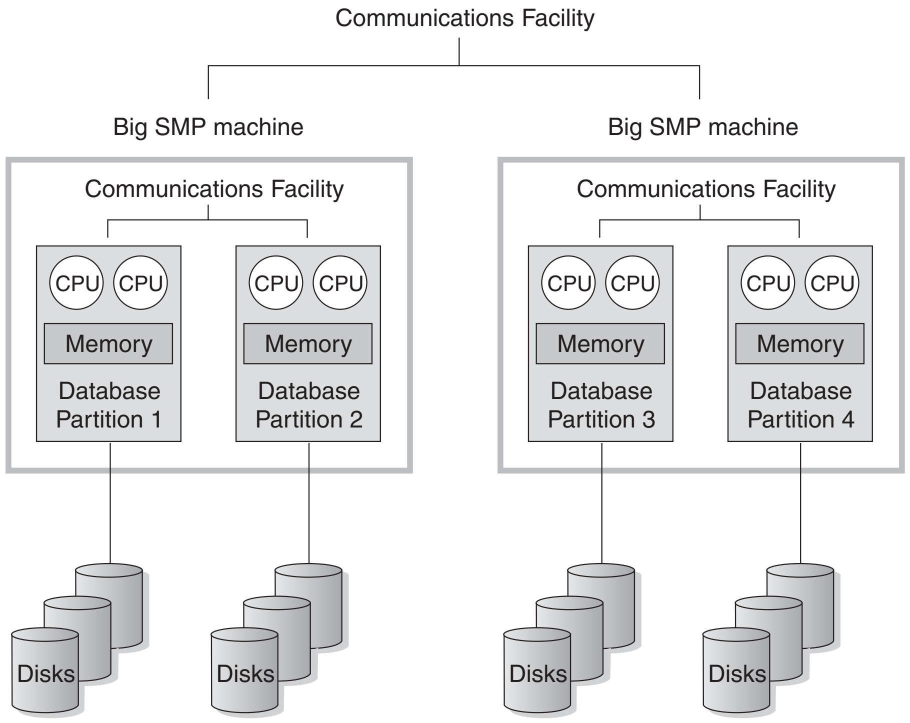


Figure 21. Partitioned Database, Symmetric Multiprocessor Systems Clustered Together

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.

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
Single Partition, Multiple Processors (SMP)	Yes	Yes	No
Multiple Partitions, One Processor (MPP)	Yes	No(1)	Yes

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**
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).			

Related concepts:

- "Parallelism" on page 26

Chapter 3. About data warehousing

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, several problems can arise when analysts access the operational data directly:

- Analysts 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, the 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*. Informational data is data that is extracted from the operational data and then transformed for decision making. For example, a data warehousing tool might copy all the sales data from the operational database, clean the data, perform calculations to summarize the data, and write the summarized data to a target in a separate database from the operational data. Users can query the separate database (the *warehouse*) without impacting the operational databases.

Data warehouse objects

The following sections describe the objects 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 define a Sales subject area and a Marketing subject area.

You then add the processes that relate to sales under the Sales subject area. Similarly, you add the definitions that relate to the marketing data under 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 the data. The sources can be almost any relational or nonrelational source (table, view, file, or predefined nickname), SAP R/3 source, or WebSphere® Site Analyzer source that has connectivity to your network.

Warehouse targets

Warehouse targets are database tables or files that contain data that has been transformed. Like a warehouse source, users can use warehouse targets to provide data to other warehouse targets. A central warehouse can provide data to departmental servers, or a main fact table in the warehouse can provide data to summary tables.

Warehouse agents and agent sites

Warehouse agents manage the flow of data between the data sources and the target warehouses. Warehouse agents are available on the AIX, Linux, iSeries, z/OS, Windows® NT, Windows 2000, and Windows XP operating systems, and for the Solaris Operating Environment. 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 to 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 workstation as the warehouse server. A remote warehouse agent is an agent that is installed on another workstation 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 workstation can have only one TCP/IP host name. However, you can define multiple agent sites on a single workstation. A logical name identifies each agent site.

The default agent site, named the Default DWC Agent Site, is a local agent that the Data Warehouse Center defines during initialization of the warehouse control database.

Processes and steps

A process contains a series of steps that perform a transformation and movement of data for a specific warehouse use. In general, a process moves source data into the warehouse. Then, the data is aggregated and summarized for warehouse use. A process can produce a single flat table or a set of summary tables. A process might also perform some specific type of data transformation.

A step is the definition of a single operation within the warehouse. By using SQL statements or calling programs, steps define how you move data and transform data. When you run a step, a transfer of data between the warehouse source and the warehouse target, or any transformation of that data, can take place.

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

- A link to its source data.
- The definition of and a link to the output table or file.
- The mechanism (either an SQL statement or a program) and definition for populating the output table or file.
- The processing options and schedule by which the output table or file is populated.

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 contains several steps. Each step performs a separate task, such as extracting the data from a database or converting it to the correct format. You might need to create several steps to completely transform and format the data and put it into its final table.

When a step or a process runs, it can affect the target in the following ways:

- Replace all the data in the warehouse target with new data
- Append the new data to the existing data
- Append a separate edition of data
- Update existing data

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 completion, either successful or not, of another step. If you schedule a process, the first step in the process runs at the scheduled time.

The following sections describe the various types of steps that you will find in the Data Warehouse Center.

SQL steps

There are two types of SQL steps. The SQL Select and Insert 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. The SQL Select and Update step uses an SQL SELECT statement to extract data from a warehouse source and update existing data in the warehouse target table.

Program steps

There are several types of program steps: DB2 for iSeries™ programs, DB2 for z/OS™ 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 a business-specific transformation that you want the Data Warehouse Center to start. Because every business has unique data transformation requirements, businesses can choose to write their own program steps or to use tools such as those provided by ETI or Vality.

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

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

Connectors

The following connectors are included with DB2 Warehouse Manger, but are purchased separately. These connectors can help you to extract data and metadata from e-business repositories.

- DB2 Warehouse Manager Connector for SAP R/3
- DB2 Warehouse Manager Connector for the Web

With the Connector for SAP R/3, you can add the extracted data to a data warehouse, transform it using the DB2 Data Warehouse Center, or analyze it using DB2 tools or other vendors' tools. With the Connector for the Web, you can bring ″clickstream″ data from IBM® WebSphere Site Analyzer into a data warehouse.

Warehouse tasks

Creating a data warehouse involves the following tasks:

- Identifying the source data (or operational data) and defining it for use as warehouse sources.
- Creating a database to use as the warehouse and defining warehouse targets.
- Defining a subject area for groups of processes that you will define in your warehouse.
- Specifying how to move and transform the source data into its format for the warehouse database by defining steps in the processes.
- Testing the steps that you define and scheduling them to run automatically.
- Administering the warehouse by defining security and monitoring database usage using the Work In Progress notebook.
- If you have DB2® Warehouse Manager, creating an information catalog of the data in the warehouse. An information catalog is a database that contains business metadata. Business metadata helps users identify and locate data and information available to them in the organization. Data Warehouse Metadata can be published to the information catalog. The information catalog can be searched to determine what data is available in the warehouse.
- 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 or measurements about the business. For example, for a manufacturing company, 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.

Part 2. Database design

Chapter 4. Logical database design

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.

Database design is not a linear process; you will probably have to redo steps as you work out the design.

What to record in a 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.

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

Table 4. Occurrences of Employee Entities and their Attributes (continued)

EMPNO	FIRSTNME	LASTNAME	WORKDEPT	JOB
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
- 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.

Related concepts:

- "Database relationships" on page 49
- "Column definitions" on page 52

Database relationships

Several types of relationships can be defined in a database. Consider the possible relationships between employees and departments.

One-to-many and many-to-one relationships

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.

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

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).

Related concepts:

- "What to record in a database" on page 47
- "Column definitions" on page 52

Column definitions

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.

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.

   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.

   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.

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.

Primary keys

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.

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 8, there appear to be many candidate keys. The EMPNO, the PHONENO, and the LASTNAME columns each uniquely identify the employee.

Table 8. EMPLOYEE Table

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

The 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.

Related concepts:

- "Identity columns" on page 56

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.

Related concepts:

- "Primary keys" on page 54

Normalization

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.
Second	Each column that is not part of the key is dependent upon the key.
Third	Each non-key column is independent of other non-key columns, and is dependent only upon the key.
Fourth	No row contains two or more independent multi-valued facts about an entity.

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 9. 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 10. Table Conforming to First Normal Form

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

Second normal form

A table is in *second normal form* if each column that is not 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 11. 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 12. 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 13. 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.

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 15). 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 14. 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 15. 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 16. 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 17. 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 18. Table Violating Fourth Normal Form

EMPNO (Primary Key)	**SKILL (Primary Key)**	**LANGUAGE (Primary Key)**
000130	Data Modelling	English
000130	Database Design	English

Table 18. Table Violating Fourth Normal Form (continued)

EMPNO (Primary Key)	SKILL (Primary Key)	LANGUAGE (Primary Key)
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 19. EMPLOYEE_SKILL Table Conforming to Fourth Normal Form

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

Table 20. 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.

Multidimensional clustering

Multidimensional clustering (MDC) provides an elegant method for flexible, continuous, and automatic clustering of data along multiple dimensions. This can result in significant improvements in the performance of queries, as well as significant reduction in the overhead of data maintenance operations such as reorganization, and index maintenance operations during insert, update and delete operations. Multidimensional clustering is primarily intended for data warehousing and large database environments, and it can also be used in online transaction processing (OLTP) environments.

Multidimensional clustering enables a table to be physically clustered on more than one key, or dimension, simultaneously. Prior to Version 8, DB2® only supported single-dimensional clustering of data, via clustering indexes. Using a clustering index, DB2 attempts to maintain the physical order of data on pages in the key order of the index, as records are inserted and updated in the table. Clustering indexes greatly improve the performance of range queries that have predicates containing the key (or keys) of the clustering index, as, with good clustering, only a portion of the table needs to be accessed, and, when the pages are sequential, more efficient prefetching can be performed.

With MDC, these benefits are extended to more than one dimension, or clustering key. In the case of query performance, range queries involving any, or any combination of, specified dimensions of the table will benefit from clustering. Not only will these queries access only those pages having records with the correct dimension values, these qualifying pages will be grouped by extents. Furthermore, although a table with a clustering index can become unclustered over time as space fills up in the table, an MDC table is able to maintain its clustering over all dimensions automatically and continuously, thus eliminating the need to reorganize the table in order to restore the physical order of the data.

When you create a table, you can specify one or more keys as dimensions along which to cluster the data. Each of these dimensions can consist of one or more columns, as index keys do. A *dimension block index* will be automatically created for each of the dimensions specified, and it will be used by the optimizer to quickly and efficiently access data along each dimension. A *composite block index* will also automatically be created, containing all dimension key columns, and will be used to maintain the clustering of data over insert and update activity. A composite block index will only be created if a single dimension does not already contain all the dimension key columns. The composite block index can also be used by the optimizer to efficiently access data having particular dimension values.

In an MDC table, every unique combination of dimension values form a logical *cell*, which is physically comprised of blocks of pages, where a block is a set of consecutive pages on disk. The set of blocks that contain pages with data having a certain key value of one of the dimension block indexes is called a *slice*. Every page of the table is part of exactly one block, and all blocks of the table consist of the same number of pages: the *blocking factor*. The blocking factor is equal to extent size, so that block boundaries line up with extent boundaries.

Consider an MDC table that records sales data for a national retailer. The table is clustered along the dimensions YearAndMonth and Region. Records in the table are stored in blocks, which contain an extent's worth of consecutive pages on disk. In Figure 22 on page 65, a block is represented by an rectangle, and is numbered according to the logical order of allocated extents in the table. The grid in the diagram represents the logical partitioning of these blocks, and each square represents a logical cell. A column or row in the grid represents a slice for a particular dimension. For example, all records containing the value ’South-central’ in the region column are found in the blocks contained in the slice defined by the ’South-central’ column in the grid. In fact, each block in this slice also only contains records having ’South-central’ in the region field. Thus, a block is contained in this slice or column of the grid if and only if it contains records having ’South-central’ in the region field.

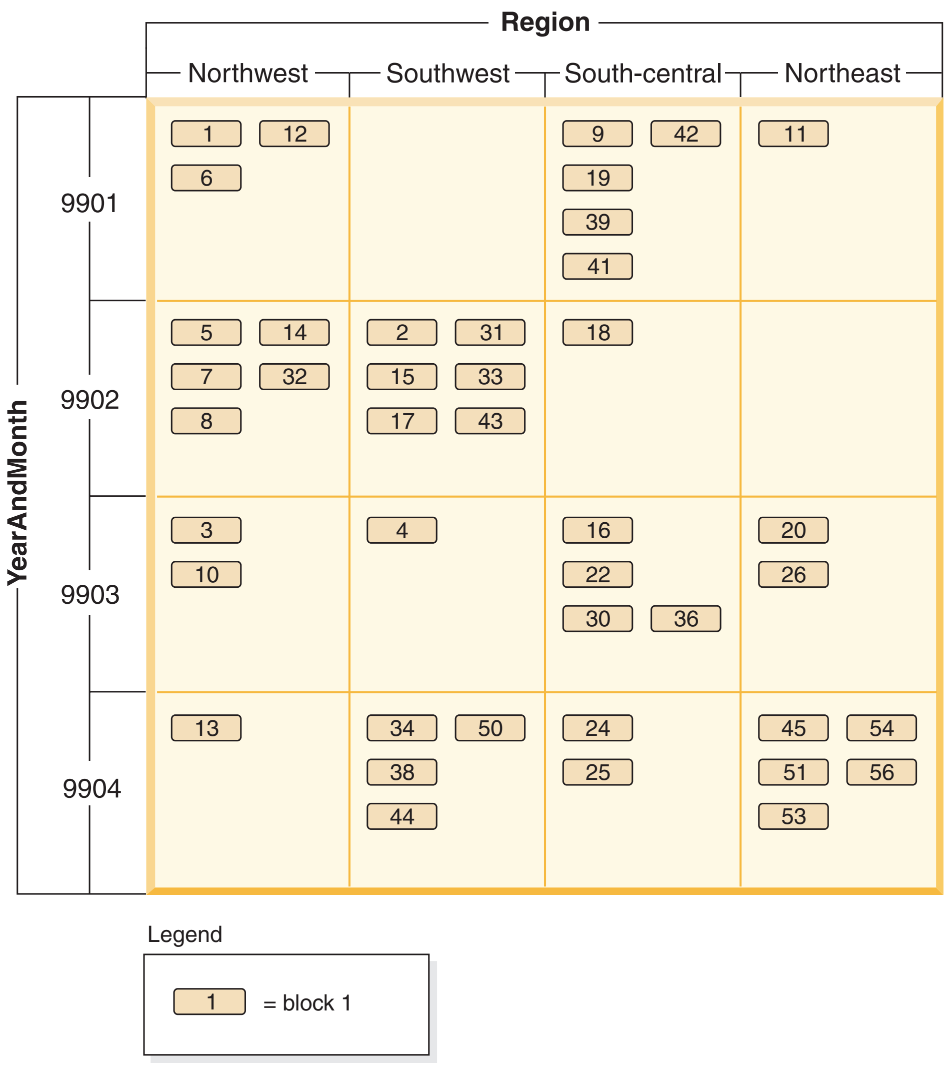


Figure 22. Multidimensional table with dimensions of Region and YearAndMonth

To facilitate the determination of which blocks comprise a slice, or which blocks contain all records having a particular dimension key value, a dimension block index is automatically created for each dimension when the table is created.

In Figure 23 on page 66, a dimension block index is created on the YearAndMonth dimension, and another on the Region dimension. Each dimension block index is structured in the same manner as a traditional RID index, except that at the leaf level the keys point to a block identifier (BID) instead of a record identifier (RID). Since each block contains potentially many pages of records, these block indexes are much smaller than RID indexes and need only be updated as new blocks are needed and therefore added to a cell, or existing blocks are emptied and therefore removed from a cell.

A slice, or the set of blocks containing pages with all records having a particular key value in a dimension, will be represented in the associated dimension block index by a BID list for that key value.

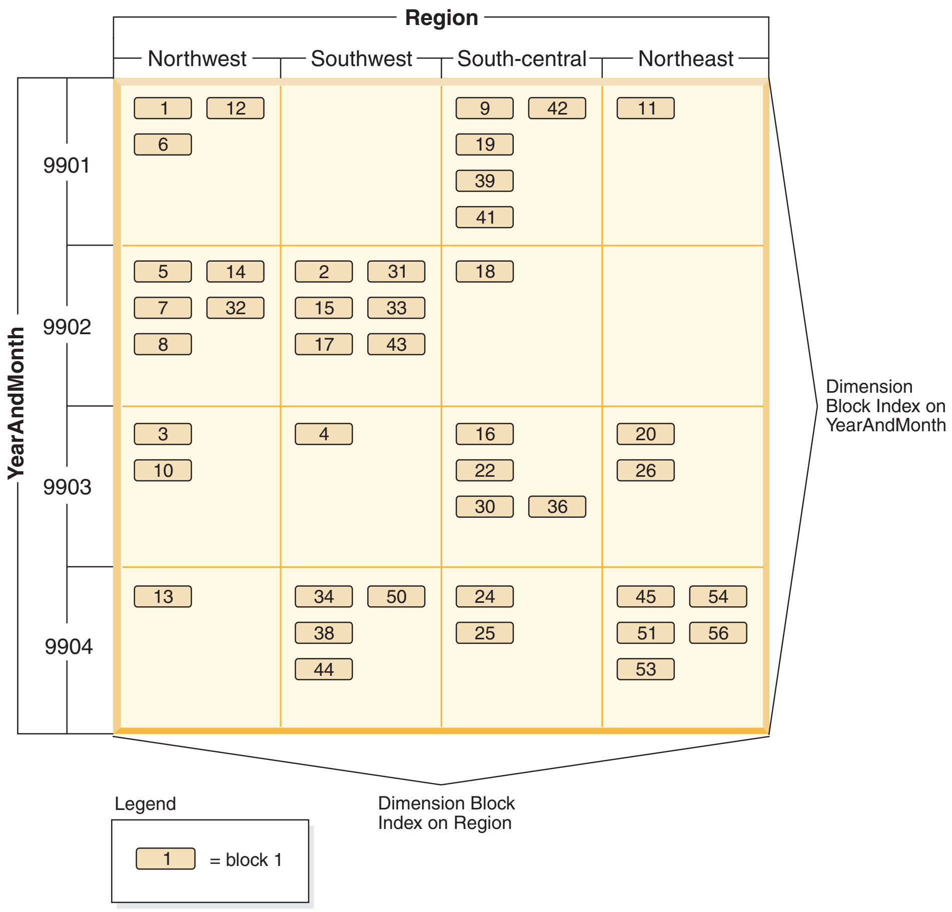


Figure 23. Multidimensional table with dimensions of Region and YearAndMonth showing dimension block indexes

Figure 24 on page 67 shows how a key from the dimension block index on Region would appear. The key is comprised of a key value, namely 'South-central', and a list of BIDs. Each BID contains a block location. In Figure 24 on page 67, the block numbers listed are the same that are found in the 'South-central' slice found in the grid for the Sales table (see Figure 22 on page 65).

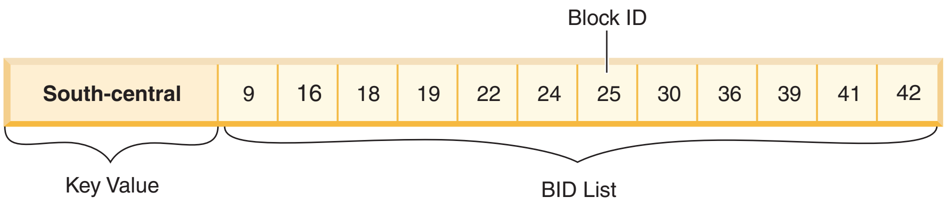


Figure 24. Key from the dimension block index on Region

Similarly, to find the list of blocks containing all records having '9902' for the YearAndMonth dimension, we would look up this value in the YearAndMonth dimension block index, shown in Figure 25.

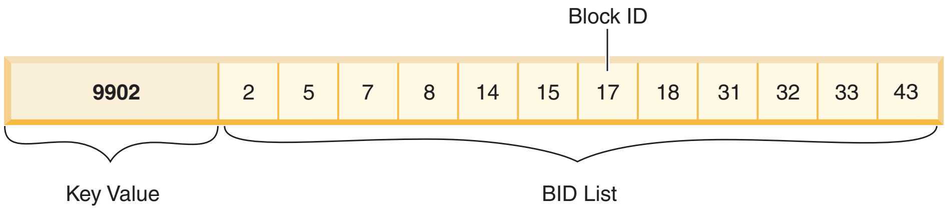


Figure 25. Key from the dimension block index on YearAndMonth

When a record is inserted into the Sales table, DB2 will determine if a cell exists for its dimension values. If one does, DB2 will insert the record into an existing block of that cell if it can, or add another block to that cell if the current blocks are full. If the cell does not yet exist, DB2 will create a new cell and add a block to it. This automatic maintenance is implemented with an additional block index, also created when the MDC table is created. This block index will be on all the dimension columns of the table, so that each key value corresponds to a particular cell in the table, and its BID list corresponds to the list of blocks comprising that cell (see Figure 26 on page 68). This type of index is a *composite block index*.

A key is found in this composite block index only for each cell of the table containing records. This block index assists in quickly and efficiently finding those blocks with records having a particular set of values for their dimensions. The composite block index is used for query processing, to access data in the table having particular dimension values. It is also used to dynamically manage and maintain the physical clustering of data along the dimensions of the table over the course of insert activity.

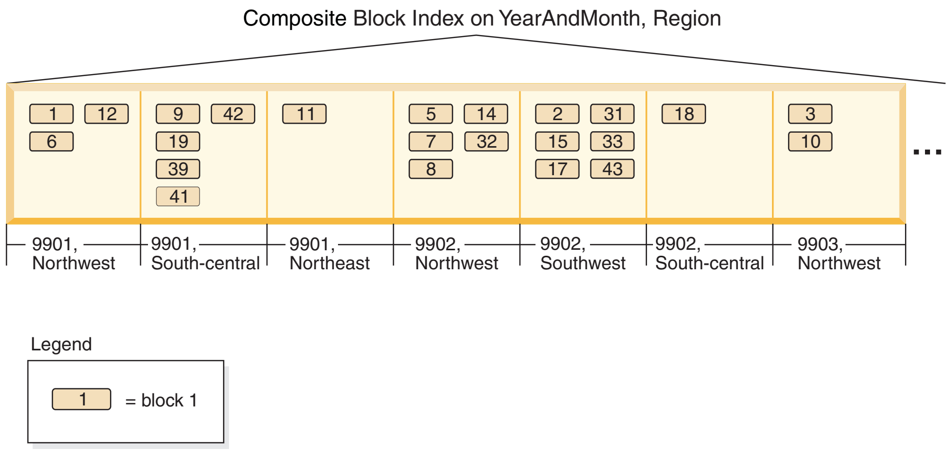


Figure 26. Composite block index on YearAndMonth, Region

For example, if you want to find all records in the Sales table having Region of 'Northwest' and YearAndMonth of '9903', DB2 would look up the key value 9903,Northwest in the composite block index, as shown in Figure 27. The key is comprised of a key value, namely '9903,Northwest', and a list of BIDs. You can see that the only BIDs listed are 3 and 10, and indeed there are only two blocks in the Sales table containing records having these two particular values.

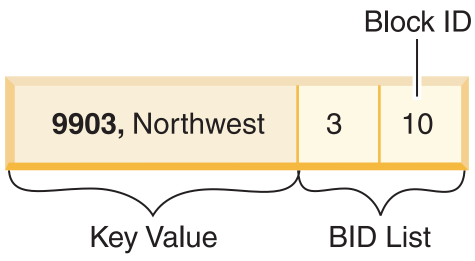
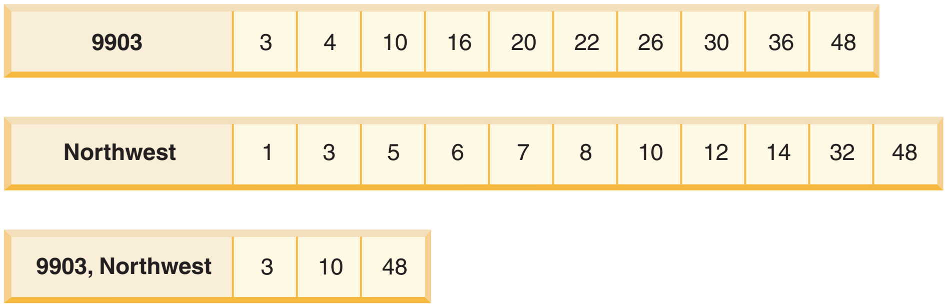


Figure 27. Key from composite block index on YearAndMonth, Region

To illustrate the use of this index during insert, take the example of inserting another record with dimension values 9903 and Northwest. DB2 would look up this key value in the composite block index and find BIDs for blocks 3 and 10. These blocks contain all records and the only records having these dimension key values. DB2 therefore inserts the new record in one of these blocks if there is space on any of their pages. If there is no space on any pages in these blocks, DB2 allocates a new block for the table, or uses a previously emptied block in the table. Note that, in this example, block 48 is currently not in use by the table. DB2 inserts the record on a page in that block, and assigns this block to this cell by adding its BID to the composite block index and to each of the dimension block indexes. See Figure 28 for an illustration of the keys of the dimension block indexes after the addition of Block 48.



9903	3	4	10	16	20	22	26	30	36	48	
Northwest	1	3	5	6	7	8	10	12	14	32	48
9903, Northwest	3	10	48								

Figure 28. Keys from the dimension block indexes after addition of Block 48

If the Sales table is clustered on three dimensions, the individual dimension block indexes can also be used to find the set of blocks containing records which satisfy a query on a subset of all of the dimensions of the table. If the table has dimensions of YearAndMonth, Region and Product, this can be thought of as a logical cube, as illustrated in Figure 29 on page 70.

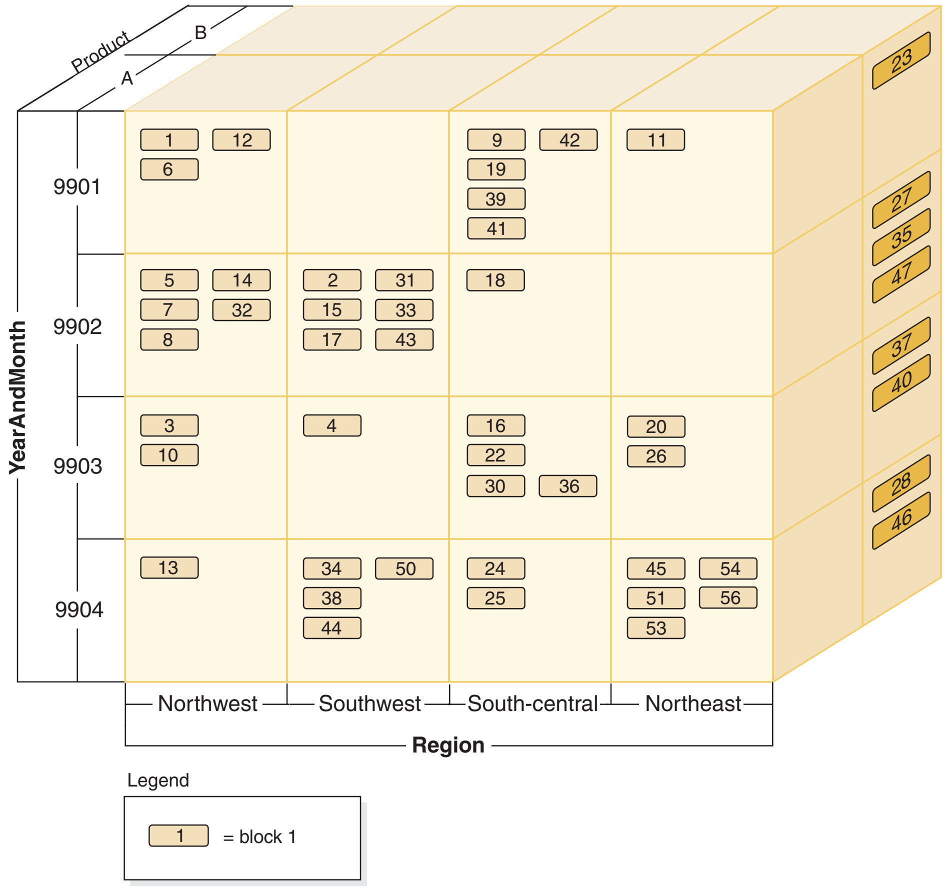


Figure 29. Multidimensional table with dimensions of Region, YearAndMonth, and Product

Four block indexes will be created on an MDC table with three dimensions: one for each of the individual dimensions, YearAndMonth, Region, and Product; and another with all of these dimension columns as its key. If you want to find all records having Product=Product A and Region=Northeast, DB2 would first determine the slice containing all blocks with Product=Product A, by looking up the Product A key in the Product dimension block index. (See Figure 30.) DB2 then determines the blocks containing all records having Region=Northeast, by looking up the Northeast key in the Region dimension block index. (See Figure 31 on page 71.)

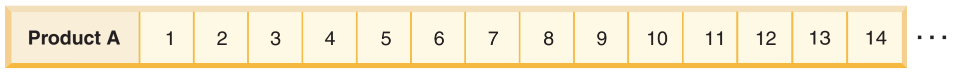


Figure 30. Key from dimension block index on Product

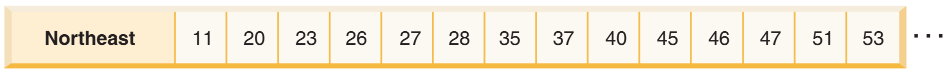


Figure 31. Key from dimension block index on Region

To find the set of blocks containing all records having both values, you have to find the intersection of these two slices. This is done by index ANDing the two BID lists. The common BID values are 11, 20, 26, 45, 54, 51, 53, and 56.

Once you have a list of blocks to scan, DB2 can do a mini-relational scan of each block. This will involve just one I/O per block, as each block is stored as an extent on disk and can be read into the buffer pool as a unit. DB2 only needs to reapply the predicates on one record in the block, as all records in the block are guaranteed to have the same dimension key values. If other predicates are present, DB2 only needs to check these on the remaining records in the block.

Regular RID-based indexes are also supported for MDC tables, and RID and block indexes can be combined by index ANDing and ORing. The index advisor will help choose RID-based indexes on an MDC table, but it will not help choose dimensions for an MDC table.

MDC tables are otherwise treated like any existing table; that is, triggers, referential integrity, views, and materialized query tables can all be defined upon them.

MDC and database partitioning:

Multidimensional clustering can be used in conjunction with database partitioning. In fact, MDC can complement database partitioning. For example, the MDC table Sales is clustered on YearAndMonth, Region, and Product. The table can also be partitioned across database partitions on the Region field (or on any other field). In the case of partitioning on Region, all blocks for a particular region will be found on the same database partition, but that partition may contain multiple region values. Each partition will contain an MDC table with those slices of the logical cube corresponding to the partition's Region partitioning values. For example, the South-central and Northwest slices may be found on one partition, while the Southwest and Northeast are found on another. Each partition will also contain each of the block indexes with key values for the blocks contained on that particular partition.

Load considerations for MDC tables:

If you roll data in to your data warehouse on a regular basis, you can use MDC tables to your advantage. In MDC tables, load will first reuse previously emptied blocks in the table before extending the table and adding new blocks for the remaining data. After you have deleted a set of data, for example, a month's worth of old data, you can use the load utility to roll in the next month's data and it can reuse the blocks that have been emptied after the (committed) deletion.

Logging considerations for MDC tables:

In cases where columns previously or otherwise indexed by RID indexes are now dimensions and so are indexed with block indexes, index maintenance and logging are significantly reduced. Only when the last record in an entire block is deleted does DB2 need to remove the BID from the block indexes and log this index operation. Similarly, only when a record is inserted to a new block (if it is the first record of a cell or an insert to a cell of currently full blocks) does DB2 need to insert a BID in the block indexes and log that operation. Since blocks can be between 2 and 256 pages of records, this block index maintenance and logging will be relatively small. Inserts and deletes to the table and to RID indexes will still be logged.

Block index considerations for MDC tables:

When you define a dimension for an MDC table, a dimension block index is created. In addition, a composite block index is created for all the dimensions that you have defined. If you have defined only one dimension for your MDC table, DB2 will create only one block index, which will serve both as the dimension block index and as the composite block index. Similarly, if you create an MDC table that has dimensions on column A, and on (column A, column B), DB2 will create a dimension block index on column A and a dimension block index on column A, column B. Because a composite block index is a block index of all the dimensions in the table, the dimension block index on column A, column B will also serve as the composite block index.

The composite block index is also used in query processing to access data in the table having particular dimension values. Note that the order of key parts in the composite block index may affect its use or applicability for query processing. The order of its key parts is determined by the order of columns found in the entire ORGANIZE BY DIMENSIONS clause used when creating the MDC table. For example, if a table is created using the statement `CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int)    ORGANIZE BY DIMENSIONS (c1, c4, (c3,c1), c2)` then the composite block index will be created on columns (c1,c4,c3,c2). Note that although c1 is specified twice in the dimensions clause, it is used only once as a key part for the composite block index, and in the order in which it is first found. The order of key parts in the composite block index makes no difference for insert processing, but may do so for query processing. Therefore, if it is more desirable to have the composite block index with column order (c1,c2,c3,c4), then the table should be created using the statement `CREATE TABLE t1 (c1 int, c2 int, c3 int, c4 int)    ORGANIZE BY DIMENSIONS (c1, c2, (c3,c1), c4)`.

Related concepts:

- "Indexes" in the *SQL Reference, Volume 1*
- "Considerations when choosing MDC table dimensions" on page 73
- "Table and index management for MDC tables" in the *Administration Guide: Performance*
- "Optimization strategies for MDC tables" in the *Administration Guide: Performance*
- "Considerations when creating MDC tables" on page 77

Related reference:

- "Lock modes for table and RID index scans of MDC tables" in the *Administration Guide: Performance*
- "Locking for block index scans for MDC tables" in the *Administration Guide: Performance*

Considerations when choosing MDC table dimensions

Queries that will benefit from MDC:

The first consideration when choosing dimensions for your table is the determination of which queries will benefit from clustering at a block level. You will want to consider creating dimension keys for columns that are involved in equality and range queries, especially for those columns with large cardinality. You will also want to consider creating dimensions for foreign keys in an MDC fact table involved in star joins with dimension tables. You should keep in mind the performance benefits of automatic and continuous clustering on more than one dimension, and of clustering at an extent or block level.

Queries that can take advantage of multidimensional clustering include queries of the following form. For all these examples, assume that there is an MDC table t1 with dimensions c1, c2, and c3.

Example 1:

```
SELECT .... FROM t1 WHERE c3 < 5000
```

This query involves a range predicate on a single dimension, so it can be internally rewritten to access the table using the dimension block index on c3.

The index will be scanned for block identifiers (BIDs) of keys having values less than 5000, and a mini-relational scan will be applied to the resulting set of blocks to retrieve the actual records.

Example 2:

```
SELECT .... FROM t1 WHERE c2 IN (1,2037)
```

This query involves an IN predicate on a single dimension, and can trigger block index based scans. This query can be internally rewritten to access the table using the dimension block index on c2. The index will be scanned for BIDs of keys having values of 1 and 2037, and a mini-relational scan will be applied to the resulting set of blocks to retrieve the actual records.

Example 3:

```
SELECT .... FROM t1
   WHERE c2 > 100 AND c1 = '16/03/1999' AND c3 > 1000 and c3 < 5000
```

This query involves range predicates on c2 and c3 and an equality predicate on c1, along with an AND operation. This can be internally rewritten to access the table on each of the dimension block indexes. A scan of the c2 block index will be done to find BIDs of keys having values greater than 100; a scan of the c3 block index will be done to find BIDs of keys having values between 1000 and 5000; and a scan of the c1 block index will be done to find BIDs of keys having the value ’16/03/1999’. Index ANDing will then be done on the resulting BIDs from each block scan, to find their intersection, and a mini-relational scan will be applied to the resulting set of blocks to find the actual records.

Example 4:

```
SELECT .... FROM t1 WHERE c1 < 5000 OR c2 IN (1,2,3)
```

This query involves a range predicate on the c1 dimension and a IN predicate on the c2 dimension, as well as an OR operation. This can be internally rewritten to access the table on the dimension block indexes c1 and c2. A scan of the c1 dimension block index will be done to find values less than 5000 and another scan of the c2 dimension block index will be done to find values 1, 2, and 3. Index ORing will be done on the resulting BIDs from each block index scan, then a mini-relational scan will be applied to the resulting set of blocks to find the actual records.

Example 5:

```
SELECT .... FROM t1 WHERE c1 = 15 AND c4 < 12
```

This query involves an equality predicate on the c1 dimension and another range predicate on a column that is not a dimension, along with an AND operation. This can be internally rewritten to access the dimension block index on c1, to get the list of blocks from the slice of the table having value 15 for c1. If there is a RID index on c4, an index scan can be done to retrieve the RIDs of records having c4 less than 12, and then the resulting list of blocks can be ANDed with this list of records. This intersection will eliminate RIDs not found in the blocks having c1 of 15, and only those listed RIDs found in the blocks that qualify will be retrieved from the table.

If there is no RID index on c4, then the block index can be scanned for the list of qualifying blocks, and during the mini-relational scan of each block, the predicate c4 < 12 can be applied to each record found.

Example 6:

```
SELECT .... FROM t1 WHERE c1 < 5 OR c4 = 100
```

This query involves a range predicate on dimension c1 and an equality predicate on a non-dimension column c4, as well as an OR operation. If there is a RID index on the c4 column, this may be internally rewritten to do index ORing of the dimension block index on c1 and the RID index on c4. If there is no index on c4, a table scan may be chosen instead, since all records must be checked. The index ORing would involve a block index scan on c1 for values less than 4, as well as a RID index scan on c4 for values of 100. A mini-relational scan would be performed on each block that qualifies, as all records within them will qualify, and any additional RIDs for records outside of these blocks will be retrieved as well.

Example 7:

```
SELECT .... FROM t1,d1,d2,d3
   WHERE t1.c1 = d1.c1 and d1.region = 'NY'
      AND t2.c2 = d2.c3 and d2.year='1994'
      AND t3.c3 = d3.c3 and d3.product='basketball'
```

This query involves a star join. In this example, t1 is the fact table and it has foreign keys c1, c2, and c3, corresponding to the primary keys of d1, d2, and d3, the dimension tables. The dimension tables do not have to be MDC tables. Region, year, and product are columns of the respective dimension tables which can be indexed using regular or block indexes (if the dimension tables are MDC tables). When we access the fact table on c1, c2, and c3 values, block index scans of the dimension block indexes on these columns can be done, and then index ANDing of the resulting BIDs. When there is a list of blocks, a mini-relational scan can be done on each block to get the records.

Density of cells:

An additional consideration when designing a multidimensional table is the expected density of cells in the table, based on present and anticipated data.

You can choose a set of dimensions, based on query performance, that cause the potential number of cells in the table to be very large, based on the number of possible values for each of the dimensions. The number of possible cells in the table is equal to the Cartesian product of the cardinalities of each of the dimensions. For example, if you cluster the table on dimensions Day, Region and Product and the data covers 5 years, you might have `1821 days * 12 regions * 5 products = 109,260` different possible cells in the table. Any cell that contains only a few records will still require an entire block of pages allocated to it, in order to store the records for that cell. If the block size is large, this table could end up being much larger than it really needs to be.

To ensure that your MDC table is not larger than it needs to be, you have several options. The first is to reduce the size of each block, by making the extent size appropriately small. In this case, each cell that contains only one block with few records on it will then contain a smaller block and less space will be wasted. The trade-off, however, is that for those cells having many records, more blocks will be needed to contain them. This increases the number of block identifiers (BIDs) for these cells in the block indexes, making these indexes larger and potentially resulting in more inserts and deletes to these indexes as blocks are more quickly emptied and filled. It also results in more small groupings of clustered data in the table for these more populated cell values, versus a smaller number of larger groupings of clustered data.

The second factor to consider in order to ensure an appropriate size for your MDC table is to consider rolling up one or more dimensions to a coarser granularity, in order to give it a lower cardinality. For example, you can continue to cluster the data in the previous example on Region and Product, but replace the dimension of Day with a dimension of YearAndMonth. This gives cardinalities of 60, 12 and 5 for YearAndMonth, Region, and Product, with a possible number of cells of 3600. Each cell will then hold a greater range of values and be less likely to contain only a few records. You should also take into account predicates commonly used on the columns involved, such as whether many are on month of date, or quarter, or day. This will affect the desirability of changing the granularity of the dimension. If, for example, most predicates are on particular days and you have clustered the table on month, DB2® will be able to use the block index on YearAndMonth to quickly narrow down which months contain the days desired and access only those associated blocks. When scanning the blocks, however, the day predicate will have to be applied to determine which days qualify. However, if you cluster on day (and day has high cardinality), the block index on day can be used to determine which blocks to scan, and the day predicate will only have to be reapplied to the first record of each cell that qualifies. In this case, it may be better to consider rolling up one of the other dimensions to increase the density of cells, as in rolling up the Region column, which contains 12 different values, to Regions West, North, South and East, using a user-defined function.

Related concepts:

- “Multidimensional clustering” on page 62

Considerations when creating MDC tables

Moving data from an existing table to an MDC table:

To move data from an existing table to an MDC table, export your data, drop the original table, create an MDC table (using the CREATE TABLE statement with the ORGANIZE BY DIMENSIONS clause) and load the MDC table with your data.

MDC tables in SMS table spaces:

In an SMS table space, a file is extended one page at a time by default. With multidimensional clustering, data is clustered in blocks, and the blocking factor is equal to the extent size of the table space. When an MDC table is extended, it is extended by a block, or extent, of pages at a time. Therefore, if you will be storing multidimensional tables in an SMS table space, it is important to enable multipage file allocation. When multipage file allocation is enabled, the SMS file is extended as a whole by exactly the size required, thereby improving performance significantly.

Run the *db2empfa* utility to enable multipage file allocation. Once multipage file allocation is enabled, it cannot be disabled.

MDC table with three dimensions:

If you know that certain predicates will be heavily used in queries, you can cluster the table on the columns involved, using the ORGANIZE BY DIMENSIONS clause.

Example 1:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
   ORGANIZE BY DIMENSIONS (c1, c3, c4)
```

The table in Example 1 is clustered on the values within three native columns forming a logical cube (that is, having three dimensions). The table can now be logically sliced up during query processing on one or more of these dimensions such that only the blocks in the appropriate slices or cells will be processed by the relational operators involved. Note that the size of a block, that is, the number of pages in a block, will be the extent size of the table.

MDC table with dimensions on more than one column:

Each dimension can be made up of one or more columns. Thus, you can create a table that is clustered on a dimension containing two columns.

Example 2:

```
CREATE TABLE T1 (c1 DATE, c2 INT, c3 INT, c4 DOUBLE)
   ORGANIZE BY DIMENSIONS (c1, (c3, c4))
```

In Example 2, the table will be clustered on two dimensions, c1 and (c3, c4). Thus, in query processing, the table can be logically sliced up on either the c1 dimension, or on the composite (c3, c4) dimension. The table will have the same number of blocks as the table in Example 1, but one less dimension block index. In Example 1, there will be three dimension block indexes, one for each of the columns c1, c3, and c4. In Example 2, there will be two dimension block indexes, one on the column c1 and the other on the columns c3 and c4. The main differences between these two approaches is that, in Example 1, queries involving just c4 can use the dimension block index on c4 to quickly and directly access blocks of relevant data. In Example 2, c4 is a second key part in a dimension block index, so queries involving just c4 involve more processing. However, in Example 2 DB2® will have one less block index to maintain and store.

MDC table with column expressions as dimensions:

Column expressions can also be used for clustering dimensions. The ability to cluster on column expressions is useful for rolling up dimensions to a coarser granularity, such as rolling up an address to a geographic location or region, or rolling up a date to a week, month, or year. In order to implement the rolling up of dimensions in this way, you can use generated columns. This type of column definition will allow the creation of columns using expressions that can represent dimensions. In Example 3, the statement creates a table clustered on one base column and two column expressions.

Example 3:

```
CREATE TABLE T1(c1 DATE, c2 INT, c3 INT, c4 DOUBLE,
   c5 DOUBLE GENERATED ALWAYS AS (c3 + c4),
   c6 GENERATED ALWAYS AS (MONTH(C1))
      ORGANIZE BY DIMENSIONS (c2, c5, c6)
```

In Example 3, column c5 is an expression based on columns c3 and c4, while column c6 rolls up column c1 to a coarser granularity in time. This statement will cluster the table based on the values in columns c2, c5, and c6.

Monotonicity:

A column expression can include any valid expression, including user-defined functions (UDFs). However, for range queries to be applied on a dimension, that is, for range scans to be possible on the underlying dimension block index, the expression must be monotonic in nature. A non-monotonic expression will only allow equality predicates to be applied to that dimension. A good example of a non-monotonic function is MONTH( ) as seen in the definition of column c6 in Example 3. If the c1 column is a date, timestamp, or valid string representation of a date or timestamp, then the function returns an integer value in the range of 1-12. Even though the output of the function is deterministic, it actually produces output similar to a step function (that is, a cyclic pattern):

```
MONTH(date('99/01/05')) = 1
MONTH(date('99/02/08')) = 2
MONTH(date('99/03/24')) = 3
MONTH(date('99/04/30')) = 4
...
MONTH(date('99/12/09')) = 12
MONTH(date('00/01/18')) = 1
MONTH(date('00/02/24')) = 2
...
```

Thus, although date in this example is continually increasing, MONTH(date) is not. More specifically, it is not guaranteed that whenever date1 is larger than date2, MONTH(date1) is greater than or equal to MONTH(date2). It is this condition that is required for monotonicity. This non-monotonicity is allowed, but it limits the dimension in that a range predicate on the base column cannot generate a range predicate on the dimension. However, a range predicate on the expression is fine, for example, `where month(c1) between 4 and 6`. This can use the index on the dimension in the usual way, with a starting key of 4 and a stop key of 6.

To make this function monotonic, you would have to include the year as the high order part of the month. DB2 provides an extension to the INTEGER built-in function to help in defining a monotonic expression on date. INTEGER(date) returns an integer representation of the date, which then can be divided to find an integer representation of the year and month. For example, `INTEGER(date('2000/05/24'))` returns `20000524`, and therefore `INTEGER(date('2000/05/24'))/100 = 200005`. The function INTEGER(date)/100 is monotonic.

Similarly, the built-in functions DECIMAL and BIGINT also have extensions so that you can derive monotonic functions. DECIMAL(timestamp) returns a decimal representation of a timestamp, and this can be used in monotonic expressions to derive increasing values for month, day, hour, minute, and so on. BIGINT(date) returns a big integer representation of the date, similar to INTEGER(date).

DB2 will determine the monotonicity of an expression, where possible, when creating the generated column for the table, or when creating a dimension from an expression in the dimensions clause. Certain functions can be recognized as monotonicity-preserving, such as DATENUM( ), DAYS( ), YEAR( ). Also, various mathematical expressions such as division, multiplication or addition of a column and a constant are monotonicity-preserving. Where DB2 determines that an expression is not monotonicity-preserving, or if it cannot determine this, the dimension will only support the use of equality predicates on its base column.

Related concepts:

- "Export Overview" in the *Data Movement Utilities Guide and Reference*
- "Multidimensional clustering" on page 62
- "Considerations when choosing MDC table dimensions" on page 73

Related tasks:

- "Defining dimensions on a table" in the *Administration Guide: Implementation*

Related reference:

- "CREATE TABLE statement" in the *SQL Reference, Volume 2*
- "db2empfa - Enable Multipage File Allocation Command" in the *Command Reference*

Constraints

A *constraint* is a rule that the database manager enforces.

There are three types of constraints:

- A *unique constraint* is a rule that forbids duplicate values in one or more columns within a table. Unique and primary keys are the supported unique constraints. For example, a unique constraint can be defined on the supplier identifier in the supplier table to ensure that the same supplier identifier is not given to two suppliers.
- A *referential constraint* is a logical rule about values in one or more columns in one or more tables. For example, a set of tables shares information about a corporation's suppliers. Occasionally, a supplier's name changes. You can define a referential constraint stating that the ID of the supplier in a table must match a supplier ID in the supplier information. This constraint prevents insert, update, or delete operations that would otherwise result in missing supplier information.
- A *table check constraint* sets restrictions on data added to a specific table. For example, a table check constraint can ensure that the salary level for an employee is at least $20,000 whenever salary data is added or updated in a table containing personnel information.

Referential and table check constraints can be turned on or off. It is generally a good idea, for example, to turn off the enforcement of a constraint when large amounts of data are loaded into a database.

Unique constraints

A *unique constraint* is the rule that the values of a key are valid only if they are unique within a table. Unique constraints are optional and can be defined in the CREATE TABLE or ALTER TABLE statement using the PRIMARY KEY clause or the UNIQUE clause. The columns specified in a unique constraint must be defined as NOT NULL. The database manager uses a unique index to enforce the uniqueness of the key during changes to the columns of the unique constraint.

A table can have an arbitrary number of unique constraints, with at most one unique constraint defined as the primary key. A table cannot have more than one unique constraint on the same set of columns.

A unique constraint that is referenced by the foreign key of a referential constraint is called the *parent key*.

When a unique constraint is defined in a CREATE TABLE statement, a unique index is automatically created by the database manager and designated as a primary or unique system-required index.

When a unique constraint is defined in an ALTER TABLE statement and an index exists on the same columns, that index is designated as unique and system-required. If such an index does not exist, the unique index is automatically created by the database manager and designated as a primary or unique system-required index.

Note that there is a distinction between defining a unique constraint and creating a unique index. Although both enforce uniqueness, a unique index allows nullable columns and generally cannot be used as a parent key.

Referential constraints

Referential integrity is the state of a database in which all values of all foreign keys are valid. A *foreign key*is a column or a set of columns in a table whose values are required to match at least one primary key or unique key value of a row in its parent table. A *referential constraint* is the rule that the values of the foreign key are valid only if one of the following conditions is true:

- They appear as values of a parent key.
- Some component of the foreign key is null.

The table containing the parent key is called the *parent table* of the referential constraint, and the table containing the foreign key is said to be a *dependent* of that table.

Referential constraints are optional and can be defined in the CREATE TABLE statement or the ALTER TABLE statement. Referential constraints are enforced by the database manager during the execution of INSERT, UPDATE, DELETE, ALTER TABLE, ADD CONSTRAINT, and SET INTEGRITY statements.

Referential constraints with a delete or an update rule of RESTRICT are enforced before all other referential constraints. Referential constraints with a delete or an update rule of NO ACTION behave like RESTRICT in most cases.

Note that referential constraints, check constraints, and triggers can be combined.

Referential integrity rules involve the following concepts and terminology:

Parent key
A primary key or a unique key of a referential constraint.

Parent row
A row that has at least one dependent row.

Parent table
A table that contains the parent key of a referential constraint. A table can be a parent in an arbitrary number of referential constraints. A table that is the parent in a referential constraint can also be the dependent in a referential constraint.

Dependent table
A table that contains at least one referential constraint in its definition. A table can be a dependent in an arbitrary number of referential constraints. A table that is the dependent in a referential constraint can also be the parent in a referential constraint.

Descendent table
A table is a descendent of table T if it is a dependent of T or a descendent of a dependent of T.

Dependent row
A row that has at least one parent row.

Descendent row
A row is a descendent of row r if it is a dependent of r or a descendent of a dependent of r.

Referential cycle
A set of referential constraints such that each table in the set is a descendent of itself.

Self-referencing table
A table that is a parent and a dependent in the same referential constraint. The constraint is called a *self-referencing constraint*.

Self-referencing row
A row that is a parent of itself.

Insert rule

The insert rule of a referential constraint is that a non-null insert value of the foreign key must match some value of the parent key of the parent table. 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.

Update rule

The update rule of a referential constraint is specified when the referential constraint is defined. The choices are NO ACTION and RESTRICT. The update rule applies when a row of the parent or a row of the dependent table is updated.

In the case of a parent row, when a value in a column of the parent key is updated, the following rules apply:

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

In the case of a dependent row, the NO ACTION update rule is implicit when a foreign key is specified. NO ACTION means that a non-null update value of a foreign key must match some value of the parent key of the parent table when the update statement is completed.

The value of a composite foreign key is null if any component of the value is null.

Delete rule

The delete rule of a referential constraint is specified when the referential constraint is defined. The choices are NO ACTION, RESTRICT, CASCADE, or SET NULL. SET NULL can be specified only if some column of the foreign key allows null values.

The delete rule of a referential constraint applies when a row of the parent table is deleted. More precisely, the rule applies when a row of the parent table is the object of a delete or propagated delete operation (defined below), and that row has dependents in the dependent table of the referential constraint. Consider an example where P is the parent table, D is the dependent table, and p is a parent row that is the object of a delete or propagated delete operation. The delete rule works as follows:

- With RESTRICT or NO ACTION, an error occurs and no rows are deleted.

- With CASCADE, the delete operation is propagated to the dependents of p in table D.
- With SET NULL, each nullable column of the foreign key of each dependent of p in table D is set to null.

Each referential constraint in which a table is a parent has its own delete rule, and all applicable delete rules are used to determine the result of a delete operation. Thus, a row cannot be deleted if it has dependents in a referential constraint with a delete rule of RESTRICT or NO ACTION, or the deletion cascades to any of its descendents that are dependents in a referential constraint with the delete rule of RESTRICT or NO ACTION.

The deletion of a row from parent table P involves other tables and can affect rows of these tables:

- If table D is a dependent of P and the delete rule is RESTRICT or NO ACTION, then D is involved in the operation but is not affected by the operation.
- If D is a dependent of P and the delete rule is SET NULL, then D is involved in the operation, and rows of D can be updated during the operation.
- If D is a dependent of P and the delete rule is CASCADE, then D is involved in the operation and rows of D can be deleted during the operation.

  If rows of D are deleted, then the delete operation on P is said to be propagated to D. If D is also a parent table, then the actions described in this list apply, in turn, to the dependents of D.

Any table that can be involved in a delete operation on P is said to be *delete-connected* to P. Thus, a table is delete-connected to table P if it is a dependent of P, or a dependent of a table to which delete operations from P cascade.

Table check constraints

A *table check constraint* is a rule that specifies the values allowed in one or more columns of every row in a table. A constraint is optional, and can be defined using the CREATE TABLE or the ALTER TABLE statement. Specifying table check constraints is done through a restricted form of a search condition. One of the restrictions is that a column name in a table check constraint on table T must identify a column of table T.

A table can have an arbitrary number of table check constraints. A table check constraint is enforced by applying its search condition to each row that is inserted or updated. An error occurs if the result of the search condition is false for any row.

When one or more table check constraints is defined in the ALTER TABLE statement for a table with existing data, the existing data is checked against the new condition before the ALTER TABLE statement completes. The SET INTEGRITY statement can be used to put the table in *check pending* state, which allows the ALTER TABLE statement to proceed without checking the data.

Related reference:
- "SET INTEGRITY statement" in the *SQL Reference, Volume 2*
- "Interaction of triggers and constraints" in the *SQL Reference, Volume 1*

Triggers

A *trigger* defines a set of actions that are performed in response to an insert, update, or delete operation on a specified table. When such an SQL operation is executed, the trigger is said to have been *activated*.

Triggers are optional and are defined using the CREATE TRIGGER statement.

Triggers can be used, along with referential constraints and check constraints, to enforce data integrity rules. Triggers can also be used to cause updates to other tables, automatically generate or transform values for inserted or updated rows, or invoke functions to perform tasks such as issuing alerts.

Triggers are a useful mechanism for defining and enforcing *transitional* business rules, which are rules that involve different states of the data (for example, a salary that cannot be increased by more than 10 percent).

Using triggers places the logic that enforces business rules inside the database. This means that applications are not responsible for enforcing these rules. Centralized logic that is enforced on all of the tables means easier maintenance, because changes to application programs are not required when the logic changes.

The following are specified when creating a trigger:
- The *subject table* specifies the table for which the trigger is defined.
- The *trigger event* defines a specific SQL operation that modifies the subject table. The event can be an insert, update, or delete operation.
- The *trigger activation time* specifies whether the trigger should be activated before or after the trigger event occurs.

The statement that causes a trigger to be activated includes a *set of affected rows*. These are the rows of the subject table that are being inserted, updated, or deleted. The *trigger granularity* specifies whether the actions of the trigger are performed once for the statement or once for each of the affected rows.

The *triggered action* consists of an optional search condition and a set of SQL statements that are executed whenever the trigger is activated. The SQL statements are only executed if the search condition evaluates to true. If the trigger activation time is before the trigger event, triggered actions can include statements that select, set transition variables, or signal SQLstates. If the trigger activation time is after the trigger event, triggered actions can include statements that select, insert, update, delete, or signal SQLstates.

The triggered action can refer to the values in the set of affected rows using *transition variables*. Transition variables use the names of the columns in the subject table, qualified by a specified name that identifies whether the reference is to the old value (before the update) or the new value (after the update). The new value can also be changed using the SET Variable statement in before, insert, or update triggers.

Another means of referring to the values in the set of affected rows is to use *transition tables*. Transition tables also use the names of the columns in the subject table, but specify a name to allow the complete set of affected rows to be treated as a table. Transition tables can only be used in after triggers, and separate transition tables can be defined for old and new values.

Multiple triggers can be specified for a combination of table, event, or activation time. The order in which the triggers are activated is the same as the order in which they were created. Thus, the most recently created trigger is the last trigger to be activated.

The activation of a trigger may cause *trigger cascading*, which is the result of the activation of one trigger that executes SQL statements that cause the activation of other triggers or even the same trigger again. The triggered actions may also cause updates resulting from the application of referential integrity rules for deletions that can, in turn, result in the activation of additional triggers. With trigger cascading, a chain of triggers and referential integrity delete rules can be activated, causing significant change to the database as a result of a single INSERT, UPDATE, or DELETE statement.

Additional 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).

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 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 materialized query table for each division or department based on the data in the original table would be helpful.

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.

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.

Chapter 5. Physical database design

After you have completed your logical database design, 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.

Database directories and files

When you create a database, information about the database including default information is stored in a directory hierarchy. The hierarchical directory structure is created for you at a location that is determined by the information you provide in the CREATE DATABASE command. If you do not specify the location of the directory path or drive when you create the database, the default location is used.

It is recommended that you explicitly state where you would like the database created.

In the directory you specify in the CREATE DATABASE command, a subdirectory that uses the name of the instance is created. This subdirectory ensures that databases created in different instances under the same directory do not use the same path. Below the instance-name subdirectory, a subdirectory named NODE0000 is created. This subdirectory differentiates partitions in a logically partitioned database environment. Below the node-name directory, a subdirectory named SQL00001 is created. This name of this subdirectory uses the database token and represents the database being created. SQL00001 contains objects associated with the first database created, and subsequent databases are given higher numbers: SQL00002, and so on. These subdirectories differentiate databases created in this instance on the directory that you specified in the CREATE DATABASE command.

The directory structure appears as follows:

```
<your_directory>/<your_instance>/NODE0000/SQL00001/
```

The database directory contains the following files that are created as part of the CREATE DATABASE command.

- The files SQLBP.1 and SQLBP.2 contain buffer pool information. Each file has a duplicate copy to provide a backup.

- The files SQLSPCS.1 and SQLSPCS.2 contain table space information. Each file has a duplicate copy to provide a backup.
- The SQLDBCON file contains database configuration information. Do not edit this file. To change configuration parameters, use either the Control Center or the command-line statements UPDATE DATABASE MANAGER CONFIGURATION and RESET DATABASE MANAGER CONFIGURATION.
- The DB2RHIST.ASC history file and its backup DB2RHIST.BAK contain history information about backups, restores, loading of tables, reorganization of tables, altering of a table space, and other changes to a database.

  The DB2TSCHNG.HIS file contains a history of tablespace changes at a log-file level. For each log file, DB2TSCHG.HIS contains information that helps to identify which tablespace are affected by the log file. Tablespace recovery uses information from this file to determine which log files to process during tablespace recovery. You can examine the contents of both history files in a text editor.
- The log control files, SQLOGCTL.LFH and SQLOGMIR.LFH, contain information about the active logs.

  Recovery processing uses information from this file to determine how far back in the logs to begin recovery. The SQLOGDIR subdirectory contains the actual log files.

  Note: You should ensure the log subdirectory is mapped to different disks than those used for your data. A disk problem could then be restricted to your data or the logs but not both. This can provide a substantial performance benefit because the log files and database containers do not compete for movement of the same disk heads. To change the location of the log subdirectory, change the *newlogpath* database configuration parameter.
- The SQLINSLK file helps to ensure that a database is used by only one instance of the database manager.

Additional information for SMS database directories

The SQLT* subdirectories contain the default System Managed Space (SMS) table spaces required for an operational database. Three default table spaces are created:
- SQLT0000.0 subdirectory contains the catalog table space with the system catalog tables.
- SQLT0001.0 subdirectory contains the default temporary table space.
- SQLT0002.0 subdirectory contains the default user data table space.

Each subdirectory or container has a file created in it called SQLTAG.NAM. This file marks the subdirectory as being in use so that subsequent table space creation does not attempt to use these subdirectories.

In addition, a file called SQL*.DAT stores information about each table that the subdirectory or container contains. The asterisk (*) is replaced by a unique set of digits that identifies each table. For each SQL*.DAT file there might be one or more of the following files, depending on the table type, the reorganization status of the table, or whether indexes, LOB, or LONG fields exist for the table:

- SQL*.BMP (contains block allocation information if it is an MDC table)
- SQL*.LF (contains LONG VARCHAR or LONG VARGRAPHIC data)
- SQL*.LB (contains BLOB, CLOB, or DBCLOB data)
- SQL*.LBA (contains allocation and free space information about SQL*.LB files)
- SQL*.INX (contains index table data)
- SQL*.DTR (contains temporary data for a reorganization of an SQL*.DAT file)
- SQL*.LFR (contains temporary data for a reorganization of an SQL*.LF file)
- SQL*.RLB (contains temporary data for a reorganization of an SQL*.LB file)
- SQL*.RBA (contains temporary data for a reorganization of an SQL*.LBA file)

Related concepts:

- "Comparison of SMS and DMS table spaces" on page 136
- "DMS device considerations" in the *Administration Guide: Performance*
- "SMS table spaces" in the *Administration Guide: Performance*
- "DMS table spaces" in the *Administration Guide: Performance*
- "Illustration of the DMS table-space address map" in the *Administration Guide: Performance*
- "Understanding the Recovery History File" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

- "CREATE DATABASE Command" in the *Command Reference*

Space requirements for database objects

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.

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
- User Table Data
- Long Field Data
- Large Object (LOB) Data
- Index Space
- Log File Space
- Temporary Work Space

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

Related concepts:

- "Space requirements for system catalog tables" on page 93
- "Space requirements for user table data" on page 94
- "Space requirements for long field data" on page 95
- "Space requirements for large object data" on page 96
- "Space requirements for indexes" on page 97
- "Space requirements for log files" on page 100
- "Space requirements for temporary tables" on page 101

Related reference:

- "db2look - DB2 Statistics and DDL Extraction Tool Command" in the *Command Reference*

Space requirements for 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.

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.

Related concepts:

- "Space requirements for database objects" on page 92
- "Definition of system catalog tables" in the *Administration Guide: Implementation*

Space requirements for user table data

By default, table data is stored on 4 KB pages. Each page (regardless of page size) contains 68 bytes of overhead for the database manager. This leaves 4028 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.

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.

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

```
   ROUND DOWN(4028/(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, 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.

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.

Related concepts:
- "Space requirements for database objects" on page 92

Space requirements for long field data

Long field data is stored in a separate table object that is structured differently than the storage space for other data types.

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.

Related concepts:

- "Space requirements for database objects" on page 92

Space requirements for large object data

Large Object (LOB) data is stored in two separate table objects that are structured differently than the storage space for other data types.

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.

Related concepts:

- “Space requirements for database objects” on page 92

Related reference:

- “Large objects (LOBs)” in the *SQL Reference, Volume 1*

Space requirements for indexes

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

```
(average index key size + 9) * number of rows * 2
```

where:

- The ″average index key size″ is the byte count of each column in the index key. (When estimating the average column size for VARCHAR and VARGRAPHIC columns, use an average of the current data size, plus two bytes. 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 + 9) * 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 five 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 + 7 + (5 * 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 / (9 + *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 two bytes 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 + 13 + (9 * 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 / (9 + *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
```

Related concepts:

- “Indexes” in the *SQL Reference, Volume 1*
- “Space requirements for database objects” on page 92

- "Index cleanup and maintenance" in the *Administration Guide: Performance*

Space requirements for log files

You will require 32 KB of space for log control files.

You will also need at least enough space for your active log configuration, which you can calculate as

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

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; in this calculation, *logsecond* cannot be set to *-1* (When *logsecond* is set to -1, you are requesting an infinite active log space.)
- *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.

If the database is enabled for circular logging, the result of this formula will provide a sufficient amount of disk space.

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.

If the database is enabled for infinite logging (that is, you set *logsecond* to *-1*), the *userexit* configuration parameter must be enabled, so you will have the same disk space considerations. DB2® will keep at least the number of active log files specified by *logprimary* in the log path, so you should not use the value of -1 for *logsecond* in the above formula. Ensure you provide extra disk space to allow the delay caused by archiving log files.

If you are mirroring the log path, you will need to double the estimated log file space requirements.

Related concepts:

- "Space requirements for database objects" on page 92
- "Understanding Recovery Logs" in the *Data Recovery and High Availability Guide and Reference*
- "Log Mirroring" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

- "Size of Log Files configuration parameter - logfilsiz" in the *Administration Guide: Performance*
- "Number of Primary Log Files configuration parameter - logprimary" in the *Administration Guide: Performance*
- "Number of Secondary Log Files configuration parameter - logsecond" in the *Administration Guide: Performance*
- "Mirror Log Path configuration parameter - mirrorlogpath" in the *Administration Guide: Performance*

Space requirements for temporary tables

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.

Related concepts:

- "Space requirements for database objects" on page 92

Related reference:

- "sqlbmtsq - Table Space Query" in the *Administrative API Reference*

Database partition groups

A database partition group is a set of one or more database partitions. When you want to create tables for the database, you first create the database partition group where the table spaces will be stored, then you create the table space where the tables will be stored.

You can define named subsets of one or more database partitions in a database. Each subset you define is known as a *database partition group*. Each subset that contains more than one database partition is known as a *multipartition database partition group*. Multipartition database partition groups can only be defined with database partitions that belong to the same instance.

Figure 32 on page 103 shows an example of a database with five partitions in which:

- A database partition group spans all but one of the database partitions (Database Partition Group 1).
- A database partition group contains one database partition (Database Partition Group 2).
- A database partition group contains two database partitions. (Database Partition Group 3).
- The database partition within Database Partition Group 2 is shared (and overlaps) with Database Partition Group 1.
- There is a single database partition within Database Partition Group 3 that is shared (and overlaps) with Database Partition Group 1.

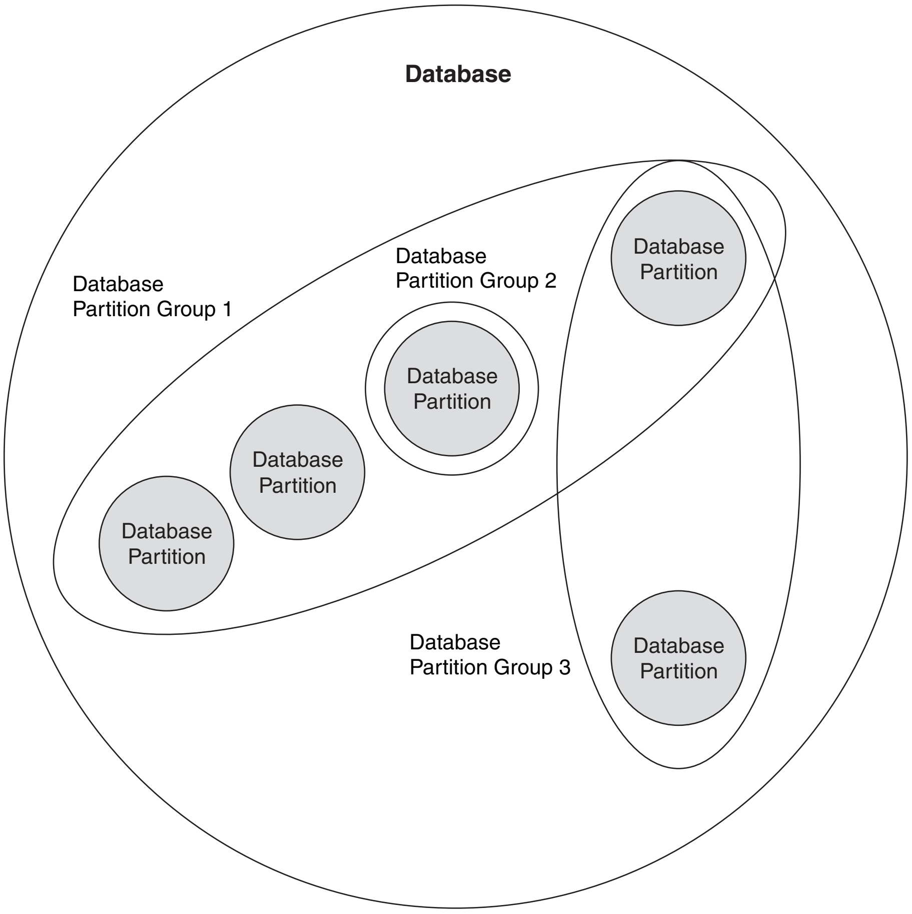


Figure 32. Database partition groups in a database

You create a new database partition group using the CREATE DATABASE PARTITION GROUP statement. You can modify it using the ALTER DATABASE PARTITION GROUP statement. Data is divided across all the partitions in a database partition group, and you can add or drop one or more database partitions from a database partition group. If you are using a multipartition database partition group, you must look at several database partition group design considerations.

Each database partition that is part of the database system configuration must already be defined in a *partition configuration file* called `db2nodes.cfg`. A database partition group can contain as little as one database partition, or as much as the entire set of database partitions defined for the database system.

When a database partition group 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 database partition group will store a given row of data.

In a non-partitioned database, no partitioning key or partitioning map is required. A database partition is a part of the database, complete with user data, indexes, configuration files, and transaction logs. Default database partition groups that were created when the database was created, are used by the database manager. IBMCATGROUP is the default database partition group for the table space containing the system catalogs. IBMTEMPGROUP is the default database partition group for system temporary table spaces. IBMDEFAULTGROUP is the default database partition group 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 database partition group, but not in IBMTEMPGROUP.

Related concepts:

- "Database partition group design" on page 104
- "Partitioning maps" on page 105
- "Partitioning keys" on page 107

Related reference:

- "ALTER DATABASE PARTITION GROUP statement" in the *SQL Reference, Volume 2*
- "CREATE DATABASE PARTITION GROUP statement" in the *SQL Reference, Volume 2*

Database partition group design

There are no database partition group design considerations if you are using a non-partitioned database.

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

- In a multipartition database partition group, 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 database partition groups, and one or more multipartition database partition groups present.
- Each database partition must be assigned a unique partition number. The same database partition may be found in one or more database partition groups.

- 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 database partition groups that do not include the database partition in the IBMCATGROUP database partition group.

You should place small tables in single-partition database partition groups, 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 database partition group. Tables are considered collocated if they reside in a multipartition database partition group, 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 database partition group, 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 database partition group than on a 32-partition database partition group.

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

Related concepts:
- "Database partition groups" on page 102
- "Partitioning maps" on page 105
- "Partitioning keys" on page 107
- "Table collocation" on page 109
- "Partition compatibility" on page 110
- "Replicated materialized query tables" on page 111

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 database partition groups, or a single entry for single-partition database partition groups. For a single-partition database partition group, 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 database partition groups, the partition numbers of the database partition group 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 database partition group of this database would be:

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

If a database partition group had been created in the database using database partitions 1 and 2, the partitioning map for that database partition group 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 33 shows how the row with the partitioning key value (c1, c2, c3) is mapped to partition 2, which, in turn, references database partition n5.

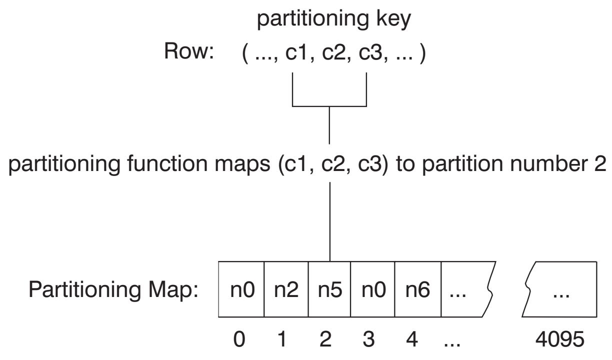

partitioning key
Row: (..., c1, c2, c3, ...)
partitioning function maps (c1, c2, c3) to partition number 2
Partitioning Map:
n0 n2 n5 n0 n6 ...
...
0 1 2 3 4 ...
4095

Figure 33. Data Distribution Using a Partitioning Map

A partitioning 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.

You can use the Get Table Partitioning Information (**sqlugtpi**) API to obtain a copy of a partitioning map that you can view.

Related concepts:

- "Database partition groups" on page 102
- "Database partition group design" on page 104
- "Partitioning keys" on page 107

Related reference:

- "sqlugtpi - Get Table Partitioning Information" in 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 database partition group, 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 database partition group, 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 database partition groups. 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 database partition group.

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 database partition group. The partitioning key for each table in a table space that is associated with a database partition group determines if the tables are collocated. Tables are considered collocated when:

- The tables are placed in table spaces that are in the same database partition group
- 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.

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 database partition group. 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.

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 database partition group is created. Each of the partition numbers is sequentially repeated in a round-robin fashion to fill the partitioning map.
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.

Related concepts:

- "Database partition groups" on page 102
- "Database partition group design" on page 104
- "Partitioning maps" on page 105

Related reference:

- "ALTER TABLE statement" in the *SQL Reference, Volume 2*

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 database partition group, and when their partitioning keys are compatible. Placing both tables in the same database partition group ensures a common partitioning map. The tables may be in different table spaces, but the table spaces must be associated with the same database partition group. The data types of the corresponding columns in each partitioning key must be *partition-compatible*.

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.

Related concepts:

- "Database partition groups" on page 102
- "Database partition group design" on page 104

- “Partitioning keys” on page 107
- “Partition compatibility” on page 110

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.

Related concepts:

- “Database partition groups” on page 102
- “Database partition group design” on page 104
- “Partitioning keys” on page 107

Replicated materialized query tables

A *materialized query table* is a table that is defined by a query that is also used to determine the data in the table. Materialized query tables can be used to improve the performance of queries. If DB2® determines that a portion of a query could be resolved using a materialized query table, the query may be rewritten by the database manager to use the materialized query table.

In a partitioned database environment, you can replicate materialized query tables. You can use *replicated materialized query tables* to improve query performance. A replicated materialized query table is based on a table that may have been created in a single-partition database partition group, but that you want replicated across all of the database partitions in the database partition group. To create the replicated materialized query table, invoke the CREATE TABLE statement with the REPLICATED keyword.

By using replicated materialized query tables, you can obtain collocation between tables that are not typically collocated. Replicated materialized query 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.

Related concepts:

- “Database partition group design” on page 104

Related tasks:

- “Creating a materialized query table” in the *Administration Guide: Implementation*

Related reference:

- “CREATE TABLE statement” in the *SQL Reference, Volume 2*

Table space design

A table space is a storage structure containing tables, indexes, large objects, and long data. Table spaces reside in database partition groups. 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 and more flexible configuration.

Since table spaces reside in database partition groups, the table space selected to hold a table defines how the data for that table is distributed across the database partitions in a database partition group. 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 34 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

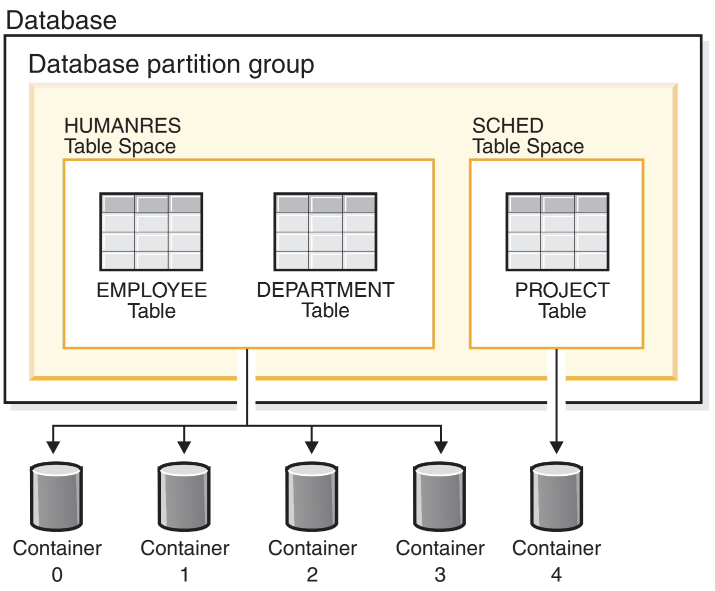


Figure 34. Table Spaces and Tables Within a Database

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 35 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.

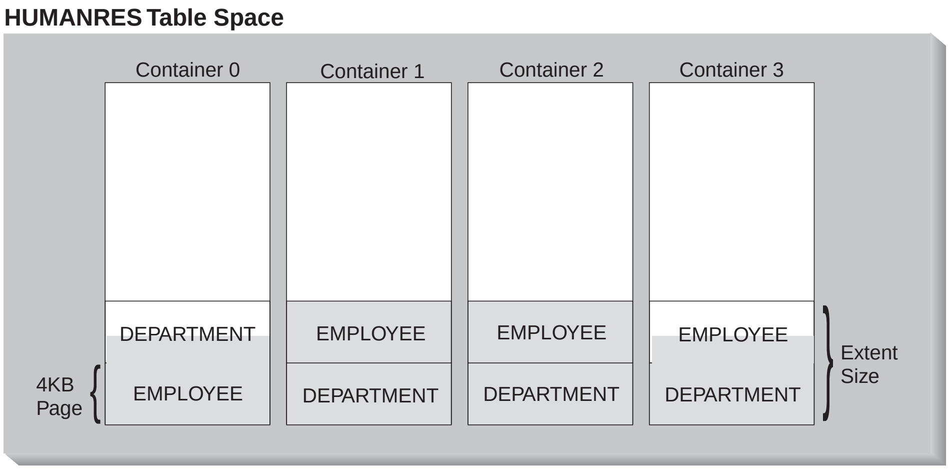


Figure 35. Containers and Extents

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 database partition group 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 database partition group 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.

  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 database partition group 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 appropriate 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. In most circumstances, it is not recommended to have more than one temporary table space of any one page size.

  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.

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, in which the operating system's file manager controls the storage space.
- Database managed space, in which the database manager controls the storage space.

Related concepts:

- "Table spaces and other storage structures" in the *SQL Reference, Volume 1*
- "System managed space" on page 115

- “Database managed space” on page 117
- “Comparison of SMS and DMS table spaces” on page 136
- “Table space disk I/O” on page 137
- “Workload considerations in table space design” on page 139
- “Extent size” on page 141
- “Relationship between table spaces and buffer pools” on page 142
- “Relationship between table spaces and database partition groups” on page 143
- “Temporary table space design” on page 144
- “Catalog table space design” on page 146

Related tasks:

- “Creating a table space” in the *Administration Guide: Implementation*
- “Optimizing table space performance when data is on RAID devices” on page 147

Related reference:

- “CREATE TABLE statement” in the *SQL Reference, Volume 2*
- “CREATE TABLESPACE statement” in the *SQL Reference, Volume 2*

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. By default, the initial table spaces created at database creation time are SMS.

Each table has at least one SMS physical file associated with it.

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. For performance reasons, if you will be storing multidimensional (MDC) tables in your SMS table space, you should enable multipage file allocation. 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.

SMS table spaces are defined 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 database partition group 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. There are also SQL limits on the size of database objects, which may affect the maximum size of a table space.

  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.

  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. 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 have the same amount of space available to 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.

Related concepts:
- "Table space design" on page 112
- "Database managed space" on page 117
- "Comparison of SMS and DMS table spaces" on page 136

Related reference:
- "db2empfa - Enable Multipage File Allocation Command" in the *Command Reference*

Database managed 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 or files whose space is managed by DB2. The database administrator decides which devices and files to use, and DB2® manages the space on those devices and files. The table space is essentially an implementation of a special purpose file system designed to best meet the needs of the database manager.

A DMS table space containing user defined tables and data can be defined as:
- A *regular* table space to store any table data and optionally index data
- A *large* table space to store long field or LOB data or index 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 large 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.
- By default, one extent in every container is reserved for overhead. 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.
- The minimum size of a DMS table space is five extents. Attempting to create a table space smaller than five extents will result in an error (SQL1422N).
  - 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.

Related concepts:
- "Table space design" on page 112
- "System managed space" on page 115
- "Comparison of SMS and DMS table spaces" on page 136
- "Table space maps" on page 119
- "How containers are added and extended in DMS table spaces" on page 123

Table space maps

A table space map is DB2's internal representation of a DMS table space that describes the logical to physical conversion of page locations in a table space. The following information describes why a table space map is useful, and where the information in a table space map comes from.

In a DB2® database, pages in a DMS table space are logically numbered from 0 to (N-1), where N is the number of usable pages in the table space.

The pages in a DMS table space are grouped into extents, based on the extent size, and from a table space management perspective, all object allocation is done on an extent basis. That is, a table might use only half of the pages in an extent but the whole extent is considered to be in use and owned by that object. By default, one extent is used to hold the container tag, and the pages in this extent cannot be used to hold data. However, if the DB2_USE_PAGE_CONTAINER_TAG registry variable is turned on, only one page is used for the container tag.

Because space in containers is allocated by extent, pages that do not make up a full extent will not be used. For example, if you have a 205-page container with an extent size of 10, one extent will be used for the tag, 19 extents will be available for data, and the five remaining pages are wasted.

If a DMS table space contains a single container, the conversion from logical page number to physical location on disk is a straightforward process where pages 0, 1, 2, will be located in that same order on disk.

It is also a fairly straightforward process in the case where there is more than one container and each of the containers is the same size. The first extent in the table space (containing pages 0 to (extent size - 1)) will be located in the first container, the second extent will be located in the second container, and so on. After the last container, the process repeats in a round-robin fashion, starting back at the first container.

For table spaces containing containers of different sizes, a simple round-robin approach cannot be used as it will not take advantage of the extra space in the larger containers. This is where the table space map comes in – it dictates how extents are positioned within the table space, ensuring that all of the extents in the physical containers are available for use.

Note: In the following examples, the container sizes do not take the size of the container tag into account. The container sizes are very small, and are just used for the purpose of illustration, they are not recommended container sizes. The examples show containers of different sizes within a table space, but you are advised to use containers of the same size.

Example 1:

There are 3 containers in a table space, each container contains 80 usable pages, and the extent size for the table space is 20. Each container will therefore have 4 extents (80 / 20) for a total of 12 extents. These extents will be located on disk as shown in Figure 36 on page 121.

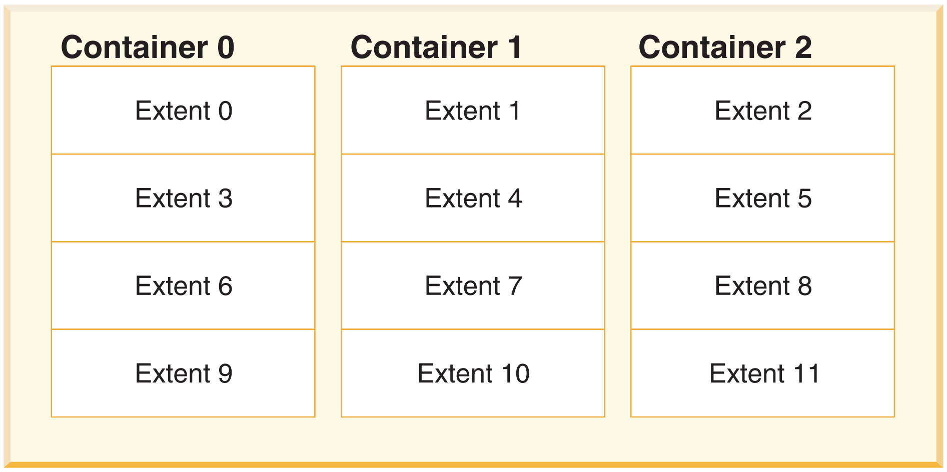


Figure 36. Table space with three containers and 12 extents

To see a table space map, take a table space snapshot using the snapshot monitor. In Example 1 where the three containers are of equal size, the table space map looks like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0      11      239        0        3    0   3 (0, 1, 2)
```

A *range* is the piece of the map in which a contiguous range of stripes all contain the same set of containers. In Example 1, all of the stripes (0 to 3) contain the same set of 3 containers (0, 1, and 2) and therefore this is considered a single range.

The headings in the table space map are Range Number, Stripe Set, Stripe Offset, Maximum extent number addressed by the range, Maximum page number addressed by the range, Start Stripe, End Stripe, Range adjustment, and Container list. These will be explained in more detail for Example 2.

This table space can also be diagrammed as shown in Figure 37 on page 122, in which each vertical line corresponds to a container, each horizontal line is called a *stripe*, and each cell number corresponds to an extent.

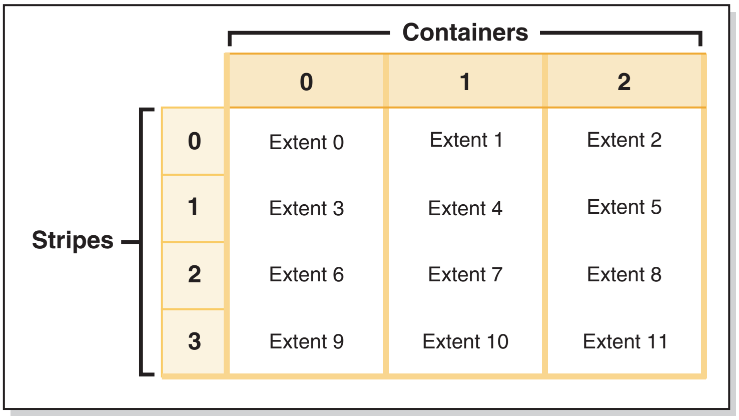


Figure 37. Table space with three containers and 12 extents, with stripes highlighted

Example 2:

There are two containers in the table space: the first is 100 pages in size, the second is 50 pages in size, and the extent size is 25. This means that the first container has four extents and the second container has two extents. The table space can be diagrammed as shown in Figure 38.

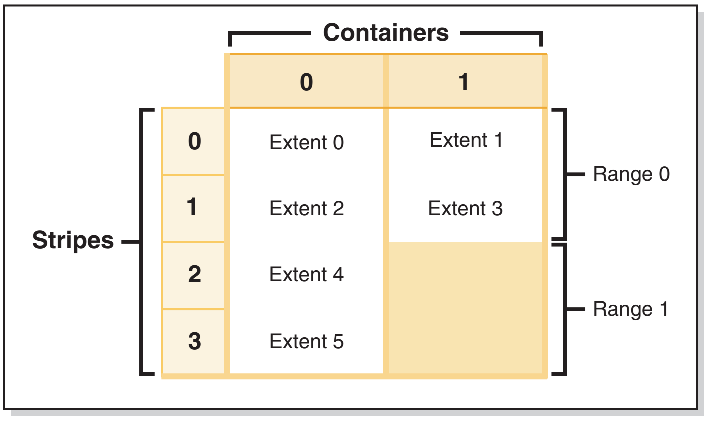


Figure 38. Table space with two containers, with ranges highlighted

Stripes 0 and 1 contain both of the containers (0 and 1) but stripes 2 and 3 only contain the first container (0). Each of these sets of stripes is a range. The table space map, as shown in a table space snapshot, looks like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       3       99        0        1    0   2 (0, 1)
   [1]       [0]       0       5      149        2        3    0   1 (0)
```

There are four extents in the first range, and therefore the maximum extent number addressed in this range (Max Extent) is 3. Each extent has 25 pages and therefore there are 100 pages in the first range. Since page numbering also starts at 0, the maximum page number addressed in this range (Max Page) is 99. The first stripe (Start Stripe) in this range is 0 and the last stripe (End Stripe) in the range is stripe 1. There are two containers in this range and those are 0 and 1. The stripe offset is the first stripe in the stripe set, which in this case is 0 because there is only one stripe set. The range adjustment (Adj.) is an offset used when data is being rebalanced in a table space. (A rebalance may occur when space is added or dropped from a table space.) While a rebalance is not taking place, this will always be 0.

There are two extents in the second range and because the maximum extent number addressed in the previous range is 3, the maximum extent number addressed in this range is 5. There are 50 pages (2 extents * 25 pages) in the second range and because the maximum page number addressed in the previous range is 99, the maximum page number addressed in this range is 149. This range starts at stripe 2 and ends at stripe 3.

Related concepts:
- "Database managed space" on page 117
- "Snapshot monitor" in the *System Monitor Guide and Reference*
- "How containers are added and extended in DMS table spaces" on page 123
- "How containers are dropped and reduced in DMS table spaces" on page 133

Related reference:
- "GET SNAPSHOT Command" in the *Command Reference*

How containers are added and extended in DMS table spaces

When a table space is created, its table space map is created and all of the initial containers are lined up such that they all start in stripe 0. This means that data will be striped evenly across all of the table space containers until the individual containers fill up. (See "Example 1" on page 125.)

The ALTER TABLESPACE statement lets you add a container to an existing table space or extend a container to increase its storage capacity.

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.

When new containers are added to a table space or existing containers are extended, a rebalance of the table space data *may* occur.

Rebalancing

The process of rebalancing when adding or extending containers involves moving table space extents from one location to another, and it is done in an attempt to keep data striped within the table space.

Access to the table space is not restricted during rebalancing; objects can be dropped, created, populated, and queried as usual. However, the rebalancing operation can have a significant impact on performance. If you need to add more than one container, and you plan on rebalancing the containers, you should add them at the same time within a single ALTER TABLESPACE statement to prevent the database manager from having to rebalance the data more than once.

The table space high-water mark plays a key part in the rebalancing process. The high-water mark is the page number of the highest allocated page in the table space. For example, a table space has 1000 pages and an extent size of 10, resulting in 100 extents. If the 42nd extent is the highest allocated extent in the table space that means that the high-water mark is 42 * 10 = 420 pages. This is not the same as used pages because some of the extents below the high-water mark may have been freed up such that they are available for reuse.

Before the rebalance starts, a new table space map is built based on the container changes made. The rebalancer will move extents from their location determined by the current map into the location determined by the new map. The rebalancer starts at extent 0, moving one extent at a time until the extent holding the high-water mark has been moved. As each extent is moved, the current map is altered one piece at a time to look like the new map. At the point that the rebalance is complete, the current map and new map should look identical up to the stripe holding the high-water mark. The current map is then made to look completely like the new map and the rebalancing process is complete. If the location of an extent in the current map is the same as its location in the new map, then the extent is not moved and no I/O takes place.

When adding a new container, the placement of that container within the new map depends on its size and the size of the other containers in its stripe set. If the container is large enough such that it can start at the first stripe in the stripe set and end at (or beyond) the last stripe in the stripe set, then it will be placed that way (see "Example 2" on page 126). If the container is not large enough to do this, it will be positioned in the map such that it ends in the last stripe of the stripe set (see "Example 4" on page 129.) This is done to minimize the amount of data that needs to be rebalanced.

Note: In the following examples, the container sizes do not take the size of the container tag into account. The container sizes are very small, and are just used for the purpose of illustration, they are not recommended container sizes. The examples show containers of different sizes within a table space, but you are advised to use containers of the same size.

Example 1:

If you create a table space with three containers and an extent size of 10, and the containers are 60, 40, and 80 pages respectively (6, 4, and 8 extents), the table space will be created with a map that can be diagrammed as shown in Figure 39.

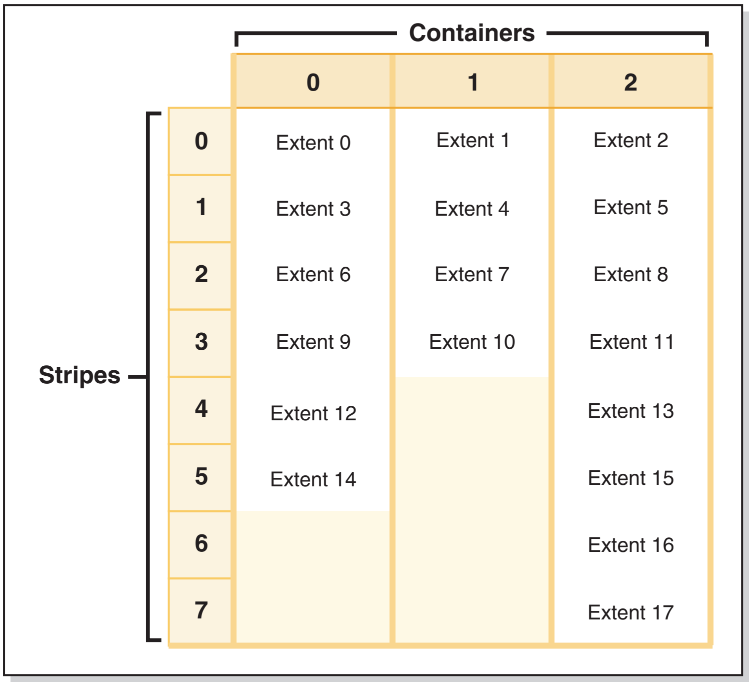


Figure 39.

The corresponding table space map, as shown in a table space snapshot, looks like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0      11      119        0        3    0   3 (0, 1, 2)
   [1]       [0]       0      15      159        4        5    0   2 (0, 2)
   [2]       [0]       0      17      179        6        7    0   1 (2)
```

The headings in the table space map are Range Number, Stripe Set, Stripe Offset, Maximum extent number addressed by the range, Maximum page number addressed by the range, Start Stripe, End Stripe, Range adjustment, and Container list.

Example 2:

If an 80-page container is added to the table space in Example 1, the container is large enough to start in the first stripe (stripe 0) and end in the last stripe (stripe 7). It is positioned such that it starts in the first stripe. The resulting table space can be diagrammed as shown in Figure 40 on page 127.

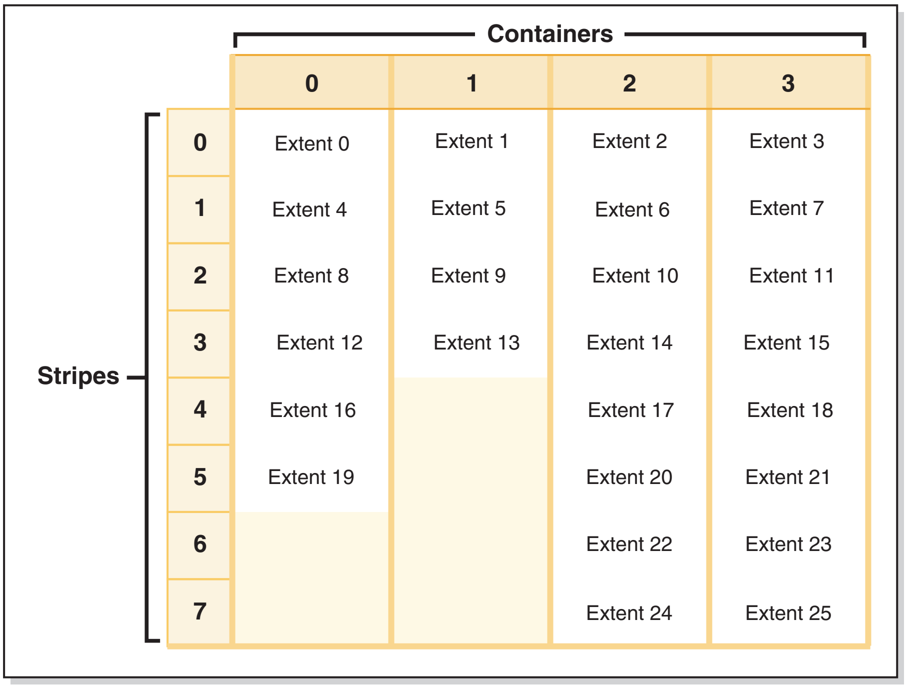


Figure 40.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range      Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number     Set     Offset  Extent     Page   Stripe   Stripe
   [0]        [0]       0      15     159        0       3     0    4 (0, 1, 2, 3)
   [1]        [0]       0      21     219        4       5     0    3 (0, 2, 3)
   [2]        [0]       0      25     259        6       7     0    2 (2, 3)
```

If the high-water mark is within extent 14, the rebalancer will start at extent 0 and will move all of the extents up to and including 14. The location of extent 0 within both of the maps is the same so this extent does not need to move. The same goes for extents 1 and 2. Extent 3 does need to move so the extent is read from the old location (second extent within container 0) and written to the new location (first extent within container 3). Every extent after this up to and including extent 14 will be moved. Once extent 14 has been moved, the current map will be made to look like the new map and the rebalancer will terminate.

If the map is altered such that all of the newly added space comes after the high-water mark, then a rebalance is not necessary and all of the space is available immediately for use. If the map is altered such that some of the space comes after the high-water mark, then the space in the stripes above the high-water mark will be available for use. The rest will not be available until the rebalance is complete.

If you decide to extend a container, the function of the rebalancer is similar. If a container is extended such that it extends beyond the last stripe in its stripe set, the stripe set will expand to fit this and the following stripe sets will be shifted out accordingly. The result is that the container will not extend into any stripe sets following it.

Example 3:

Consider the table space from Example 1. If you extend container 1 from 40 pages to 80 pages, the new table space will look like Figure 41.

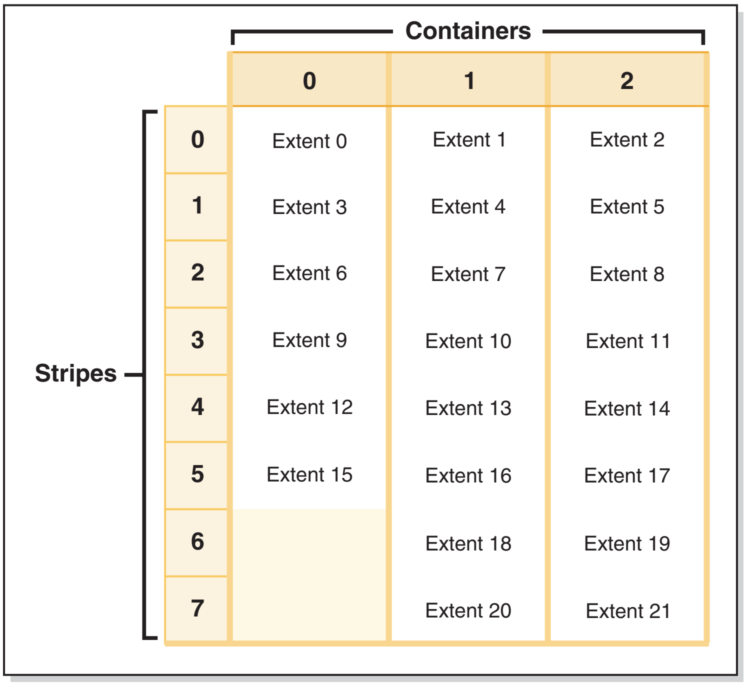


Figure 41.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range      Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number     Set     Offset  Extent     Page   Stripe   Stripe
   [0]        [0]       0       17     179        0        5     0   3 (0, 1, 2)
   [1]        [0]       0       21     219        6        7     0   2 (1, 2)
```

Example 4:

Consider the table space from "Example 1" on page 125. If a 50-page (5-extent) container is added to it, the container will be added to the new map in the following way. The container is not large enough to start in the first stripe (stripe 0) and end at or beyond the last stripe (stripe 7), so it is positioned such that it ends in the last stripe. (See Figure 42.)

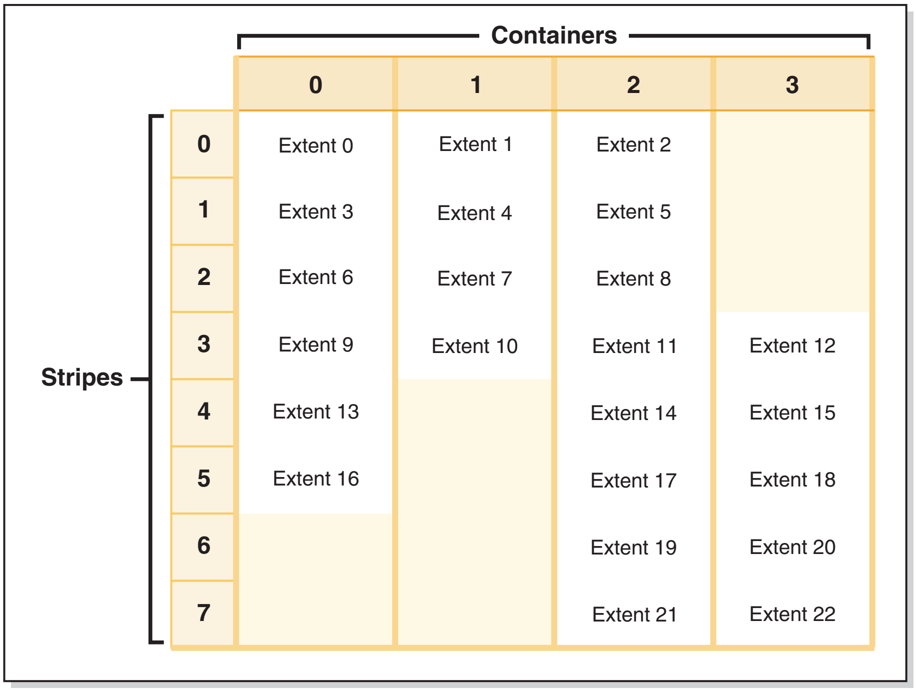


Figure 42.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       8       89        0        2    0   3 (0, 1, 2)
   [1]       [0]       0      12      129        3        3    0   4 (0, 1, 2, 3)
   [2]       [0]       0      18      189        4        5    0   3 (0, 2, 3)
   [3]       [0]       0      22      229        6        7    0   2 (2, 3)
```

To extend a container, use the EXTEND or RESIZE option on the ALTER TABLESPACE statement. To add containers and rebalance the data, use the ADD option on the ALTER TABLESPACE statement. If you are adding a container to a table space that already has more than one stripe set, you can specify which stripe set you want to add to. To do this, you use the ADD TO STRIPE SET option on the ALTER TABLESPACE statement. If you do not specify a stripe set, the default behavior will be to add the container to the current stripe set. The current stripe set is the most recently created stripe set, not the one that last had space added to it.

Any change to a stripe set may cause a rebalance to occur to that stripe set and any others following it.

You can monitor the progress of a rebalance by using table space snapshots. A table space snapshot can provide information about a rebalance such as the start time of the rebalance, how many extents have been moved, and how many extents need to move.

Without rebalancing (using stripe sets)

If you add or extend a container, and the space added is above the table space high-water mark, a rebalance will not occur.

Adding a container will almost always add space below the high-water mark. In other words, a rebalance is often necessary when you add a container. There is an option to force new containers to be added above the high-water mark, which allows you to choose not to rebalance the contents of the table space. An advantage of this method is that the new container will be available for immediate use. The option not to rebalance applies only when you add containers, not when you extend existing containers. When you extend containers you can only avoid rebalancing if the space you add is above the high-water mark. For example, if you have a number of containers that are the same size, and you extend each of them by the same amount, the relative positions of the extents will not change, and a rebalance will not occur.

Adding containers without rebalancing is done by adding a new *stripe set*. A stripe set is a set of containers in a table space that has data striped across it separately from the other containers that belong to that table space. You use a new stripe set when you want to add containers to a table space without rebalancing the data. The existing containers in the existing stripe sets remain untouched, and the containers you are adding become part of a new stripe set.

To add containers without rebalancing, use the BEGIN NEW STRIPE SET option on the ALTER TABLESPACE statement.

Example 5:

If you have a table space with three containers and an extent size of 10, and the containers are 30, 40, and 40 pages (3, 4, and 4 extents respectively), the table space can be diagrammed as shown in Figure 43.

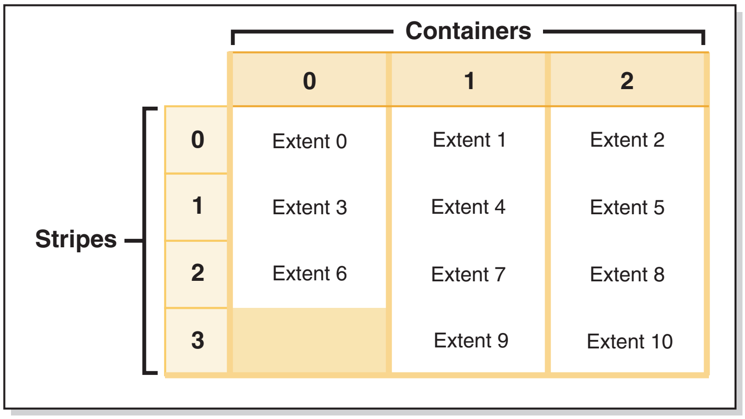


Figure 43.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range      Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number     Set     Offset  Extent     Page   Stripe   Stripe
   [0]        [0]       0        8     89        0        2     0   3 (0, 1, 2)
   [1]        [0]       0       10    109        3        3     0   2 (1, 2)
```

Example 6:

When you add two new containers that are 30 pages and 40 pages (3 and 4 extents respectively) with the BEGIN NEW STRIPE SET option, the existing ranges will not be affected and instead a new set of ranges will be created. This new set of ranges is a stripe set and the most recently created one is called the current stripe set. After the two new containers have been added, the table space will look like Figure 44.

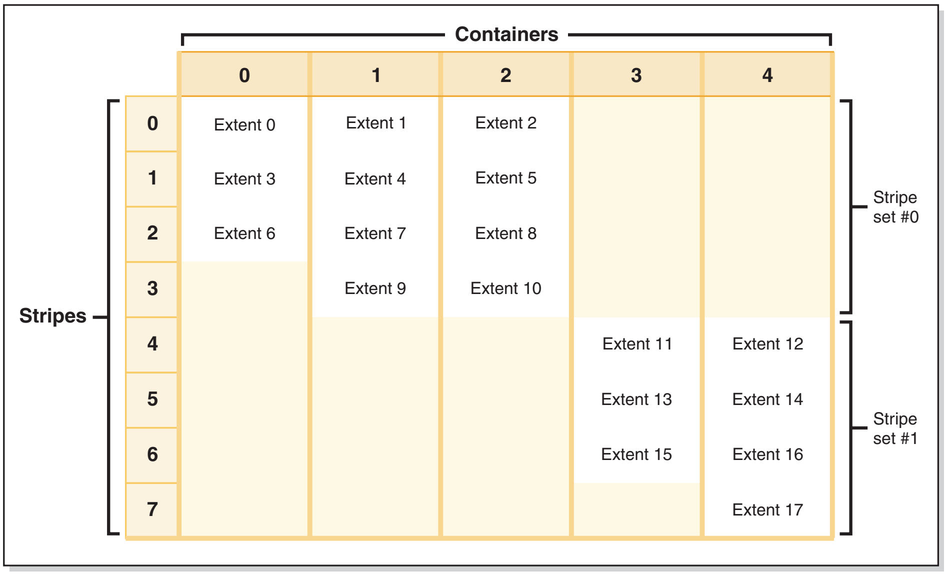


Figure 44.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range      Stripe  Stripe  Max         Max     Start    End     Adj.   Containers
Number     Set     Offset  Extent      Page    Stripe   Stripe
   [0]        [0]       0       8        89         0        2     0   3 (0, 1, 2)
   [1]        [0]       0      10       109         3        3     0   2 (1, 2)
   [2]        [1]       4      16       169         4        6     0   2 (3, 4)
   [3]        [1]       4      17       179         7        7     0   1 (4)
```

If you add new containers to a table space, and you do *not* use the TO STRIPE SET option with the ADD clause, the containers will be added to the current stripe set (the highest stripe set). You can use the ADD TO STRIPE SET clause to add containers to any stripe set in the table space. You must specify a valid stripe set.

DB2® tracks the stripe sets using the table space map, and adding new containers without rebalancing will generally cause the map to grow faster than when containers are rebalanced. When the table space map becomes too large, you will receive error SQL0259N when you try to add more containers.

Related concepts:
- "Table space maps" on page 119

Related tasks:
- "Adding a container to a DMS table space" in the *Administration Guide: Implementation*
- "Modifying containers in a DMS table space" in the *Administration Guide: Implementation*

Related reference:
- "ALTER TABLESPACE statement" in the *SQL Reference, Volume 2*
- "GET SNAPSHOT Command" in the *Command Reference*
- "Table space activity data elements" in the *System Monitor Guide and Reference*

How containers are dropped and reduced in DMS table spaces

With a DMS table space, you can drop a container from the table space or reduce the size of a container. You use the ALTER TABLESPACE statement to accomplish this.

Dropping or reducing a container will only be allowed if the number of extents being dropped by the operation is less than or equal to the number of free extents above the high-water mark in the table space. This is necessary because page numbers cannot be changed by the operation and therefore all extents up to and including the high-water mark must sit in the same logical position within the table space. Therefore, the resulting table space must have enough space to hold all of the data up to and including the high-water mark. In the situation where there is not enough free space, you will receive an error immediately upon execution of the statement.

The high-water mark is the page number of the highest allocated page in the table space. For example, a table space has 1000 pages and an extent size of 10, resulting in 100 extents. If the 42nd extent is the highest allocated extent in the table space that means that the high-water mark is 42 * 10 = 420 pages. This is not the same as used pages because some of the extents below the high-water mark may have been freed up such that they are available for reuse.

When containers are dropped or reduced, a rebalance will occur if data resides in the space being dropped from the table space. Before the rebalance starts, a new table space map is built based on the container changes made. The rebalancer will move extents from their location determined by the current map into the location determined by the new map. The rebalancer starts with the extent that contains the high-water mark, moving one extent at a time until extent 0 has been moved. As each extent is moved, the current map is altered one piece at a time to look like the new map. If the location of an extent in the current map is the same as its location in the new map, then the extent is not moved and no I/O takes place. Because the rebalance moves extents starting with the highest allocated one, ending with the first extent in the table space, it is called a *reverse rebalance* (as opposed to the *forward rebalance* that occurs when space is added to the table space after adding or extending containers).

When containers are dropped, the remaining containers are renumbered such that their container IDs start at 0 and increase by 1. If all of the containers in a stripe set are dropped, the stripe set will be removed from the map and all stripe sets following it in the map will be shifted down and renumbered such that there are no gaps in the stripe set numbers.

Note: In the following examples, the container sizes do not take the size of the container tag into account. The container sizes are very small, and are just used for the purpose of illustration, they are not recommended container sizes. The examples show containers of different sizes within a table space, but this is just for the purpose of illustration; you are advised to use containers of the same size.

For example, consider a table space with three containers and an extent size of 10. The containers are 20, 50, and 50 pages respectively (2, 5, and 5 extents). The table space diagram is shown in Figure 45.

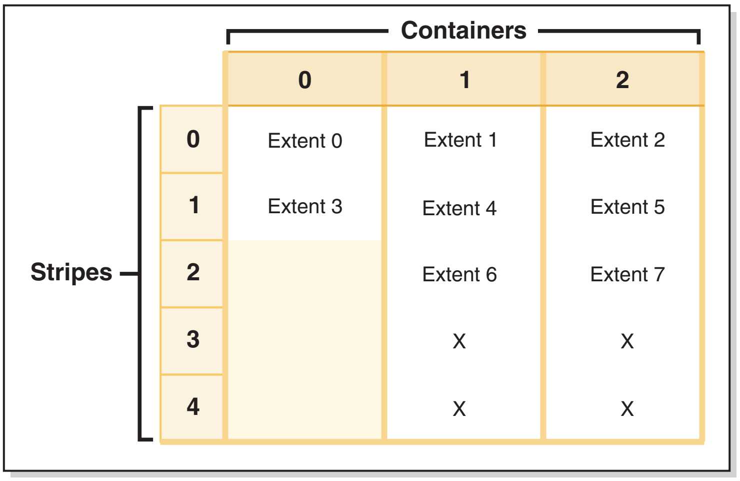


Figure 45.

An X indicates that there is an extent but there is no data in it.

If you want to drop container 0, which has two extents, there must be at least two free extents above the high-water mark. The high-water mark is in extent 7, leaving four free extents, therefore you can drop container 0.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End     Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       5      59        0        1     0    3 (0, 1, 2)
   [1]       [0]       0      11     119        2        4     0    2 (1, 2)
```

After the drop, the table space will have just Container 0 and Container 1. The new table space diagram is shown in Figure 46.

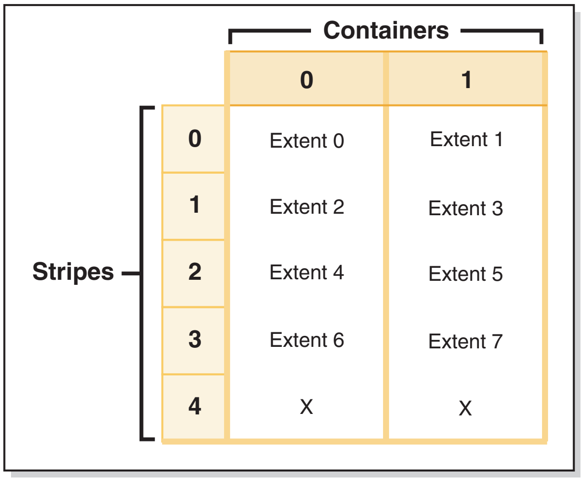


Figure 46.

The corresponding table space map, as shown in a table space snapshot, will look like this:

```
Range     Stripe  Stripe  Max        Max    Start    End     Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       9      99        0        4     0    2 (0, 1)
```

If you want to reduce the size of a container, the rebalancer works in a similar way.

To reduce a container, use the REDUCE or RESIZE option on the ALTER TABLESPACE statement. To drop a container, use the DROP option on the ALTER TABLESPACE statement.

Related concepts:

- "Table space maps" on page 119

Related tasks:

- "Modifying containers in a DMS table space" in the *Administration Guide: Implementation*

Related reference:

- "ALTER TABLESPACE statement" in the *SQL Reference, Volume 2*
- "GET SNAPSHOT Command" in the *Command Reference*
- "Table space activity data elements" in the *System Monitor Guide and Reference*

Comparison of SMS and DMS table spaces

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 table space 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 or extending containers, using the ALTER TABLESPACE statement. Existing data can be 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, 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.

Related concepts:
- "Table space design" on page 112
- "System managed space" on page 115
- "Database managed space" on page 117

Table space disk 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.

Sequential prefetching performance can be further enhanced if pages can be read from disk into contiguous pages within a buffer pool. Since buffer pools are page-based by default, there is no guarantee of finding a set of contiguous pages when reading in contiguous pages from disk. Block-based buffer pools can be used for this purpose because they not only contain a page area, they also contain a block area for sets of contiguous pages. Each set of contiguous pages is named a block and each block contains a number of pages referred to as blocksize. The size of the page and block area, as well as the number of pages in each block is configurable.

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 changing the PREFETCHSIZE option on the CREATE TABLESPACE or ALTER TABLESPACE statements. (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.

Related concepts:
- "Table space design" on page 112
- "Prefetching data into the buffer pool" in the *Administration Guide: Performance*

Related reference:
- "ALTER TABLESPACE statement" in the *SQL Reference, Volume 2*
- "CREATE TABLESPACE statement" in the *SQL Reference, Volume 2*

Workload considerations in table space design

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, often involve frequent insert or update activity and queries which usually return small sets of data. Given that the access is random, and involves one or a few pages, prefetching is less likely to occur.

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. However, setting a sufficient number of page cleaners, using the *chngpgs_thresh* configuration parameter, is important.

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. If the number of containers changes, or there is a need to make prefetching more or less aggressive, the PREFETCHSIZE value can be changed accordingly by using the ALTER TABLESPACE statement.

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 decision-support system (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.

Related concepts:
- "System managed space" on page 115
- "Database managed space" on page 117

Related reference:
- "Changed Pages Threshold configuration parameter - chngpgs_thresh" in the *Administration Guide: Performance*
- "ALTER TABLESPACE statement" in the *SQL Reference, Volume 2*
- "CREATE TABLESPACE statement" in the *SQL Reference, Volume 2*
- "SQL limits" in the *SQL Reference, Volume 1*

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. Space in SMS table spaces is allocated one extent at a time, if you have enabled multipage file allocation.

  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 metadata describing the LOB data.

  – A block map object for multidimensional tables.

  Each table object is stored separately, and each object allocates new extents as needed. Each table object is also paired with a metadata 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.
- 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.

Related concepts:
- "Table space design" on page 112

Related reference:
- "CREATE TABLESPACE statement" in the *SQL Reference, Volume 2*
- "db2empfa - Enable Multipage File Allocation Command" in the *Command Reference*

Relationship between table spaces and 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), it must have the same page size, 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 example, if you have a table space with one or more large (larger than available memory) 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.

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.

Related concepts:
- "Table spaces and other storage structures" in the *SQL Reference, Volume 1*

Related reference:
- "ALTER BUFFERPOOL statement" in the *SQL Reference, Volume 2*
- "ALTER TABLESPACE statement" in the *SQL Reference, Volume 2*
- "CREATE BUFFERPOOL statement" in the *SQL Reference, Volume 2*
- "CREATE TABLESPACE statement" in the *SQL Reference, Volume 2*

Relationship between table spaces and database partition groups

In a partitioned database environment, each table space is associated with a specific database partition group. This allows the characteristics of the table space to be applied to each partition in the database partition group. The database partition group must exist (it is defined with the CREATE DATABASE PARTITION GROUP statement), and the association between the table space and the database partition group is defined when the table space is created using the CREATE TABLESPACE statement.

You cannot change the association between table space and database partition group using the ALTER TABLESPACE statement. You can only change the table space specification for individual partitions within the database partition group. In a single-partition environment, each table space is associated with the default database partition group. The default database partition group, when defining a table space, is IBMDEFAULTGROUP, unless a system temporary table space is being defined; then IBMTEMPGROUP is used.

Related concepts:

- "Table spaces and other storage structures" in the *SQL Reference, Volume 1*
- "Database partition groups" on page 102
- "Table space design" on page 112

Related reference:

- "CREATE DATABASE PARTITION GROUP statement" in the *SQL Reference, Volume 2*
- "CREATE TABLESPACE statement" in the *SQL Reference, Volume 2*

Temporary table space design

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 directly in the target table space. Of course, this type of reorganization requires that there be extra space in the target table space for the reorganization process.
- 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 system 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:
  - There is more overhead in the creation of a temporary table when using DMS versus SMS.
  - 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.

Related concepts:
- "Table space design" on page 112
- "System managed space" on page 115

Related reference:

- "REORG INDEXES/TABLE Command" in the *Command Reference*

Catalog table space design

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 cache, 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.

Related concepts:

- "Definition of system catalog tables" in the *Administration Guide: Implementation*
- "Table space design" on page 112
- "System managed space" on page 115
- "Database managed space" on page 117

Optimizing table space performance when data is on RAID devices

This section describes how to optimize performance when data is placed on Redundant Array of Independent Disks (RAID) devices.

Procedure:

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 to enable parallel I/O for 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_PARALLEL_IO also affects table spaces with more than one container defined. If you do not set the registry variable, the I/O parallelism is equal to the number of containers in the table space. If you set the registry variable, the I/O parallelism is equal to the result of prefetch size divided by extent size. You might want to set the registry variable if the individual containers in the table space are striped across multiple physical disks.

For example, a table space has two containers and the prefetch size is four times the extent size. If the registry variable is not set, a prefetch request for this table space will be broken into two requests (each request will be for two extents). Provided that the prefetchers are available to do work, two prefetchers can be working on these requests in parallel. In the case where the registry variable is set, a prefetch request for this table space will be broken into four requests (one extent per request) with a possibility of four prefetchers servicing the requests in parallel.

In this example, if each of the two containers had a single disk dedicated to it, setting the registry variable for this table space might result in contention on those disks since two prefetchers will be accessing each of the two disks at once. However, if each of the two containers was striped across multiple disks, setting the registry variable would potentially allow access to four different disks at once.

DB2_USE_PAGE_CONTAINER_TAG:

By default, DB2 uses the first extent of each DMS container (file or device) to store a container tag. The container tag is DB2's metadata for the container. In earlier versions of DB2, the first page was used for the container tag, instead of the first extent, and as a result less space in the container was used to store the tag. (In earlier versions of DB2, the DB2_STRIPED_CONTAINERS registry variable was used to create table spaces with an extent sized tag. However, because this is now the default behavior, this registry variable no longer has any affect.)

When the DB2_USE_PAGE_CONTAINER_TAG registry variable is set to ON, any new DMS containers created will be created with a one-page tag, instead of a one-extent tag (the default). There will be no impact to existing containers that were created before the registry variable was set.

Setting this registry variable to ON is not recommended unless you have very tight space constraints, or you require behavior consistent with pre-Version 8 databases.

Setting this registry variable to ON can have a negative impact on I/O performance if RAID devices are used for table space containers. 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, if this registry variable is set to ON, a one-page container tag will be used and the extents will not line up with the RAID stripes. As a result it may be necessary during an I/O request to access more physical disks than would be optimal. Users are thus strongly advised against setting this registry variable.

To create containers with one-page container tags, set this registry variable to ON, and then stop and restart the instance:

```
db2set DB2_USE_PAGE_CONTAINER_TAG=ON
db2stop
db2start
```

To stop creating containers with one-page container tags, reset this registry variable, and then stop and restart the instance.

```
db2set DB2_USE_PAGE_CONTAINER_TAG=
db2stop
db2start
```

The Control Center, the LIST TABLESPACE CONTAINERS command, and the GET SNAPSHOT FOR TABLESPACES command do not show whether a container has been created with a page or extent sized tag. They use the label ″file″ or ″device″, depending on how the container was created. To verify whether a container was created with a page- or extent-size tag, 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.

Related concepts:

- "Table space design" on page 112

Related reference:

- "System environment variables" in the *Administration Guide: Performance*

Considerations when choosing table spaces for your tables

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 database partition group 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.

- 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.

Related concepts:

- "Database partition groups" on page 102

- “System managed space” on page 115
- “Database managed space” on page 117
- “Comparison of SMS and DMS table spaces” on page 136

Chapter 6. Designing Distributed Databases

Units of work

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.

Related reference:

- "COMMIT statement" in the *SQL Reference, Volume 2*
- "ROLLBACK statement" in the *SQL Reference, Volume 2*

Updating 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*.

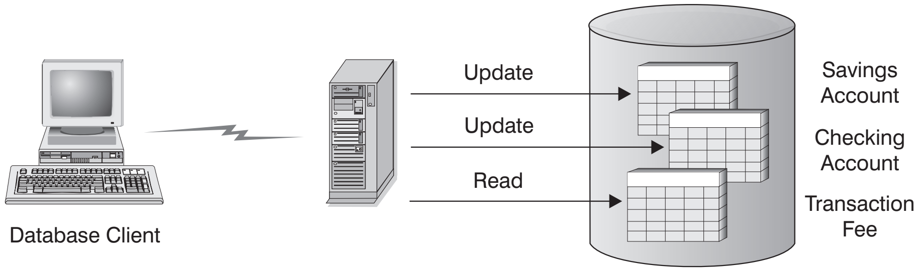


Figure 47. Using a Single Database in a Transaction

Figure 47 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).

Procedure:

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
2. If physically remote, set up the database server to use the appropriate communications protocol
3. If physically remote, catalog the node and the database to identify the database on the database server
4. Precompile your application program to specify a type 1 connection; that is, specify CONNECT 1 (the default) on the PRECOMPILE PROGRAM command.

Related concepts:

- "Units of work" on page 153

Related tasks:

- "Updating a single database in a multi-database transaction" on page 155
- "Updating multiple databases in a transaction" on page 156

Related reference:

- "PRECOMPILE Command" in the *Command Reference*

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 in a multi-database transaction

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).

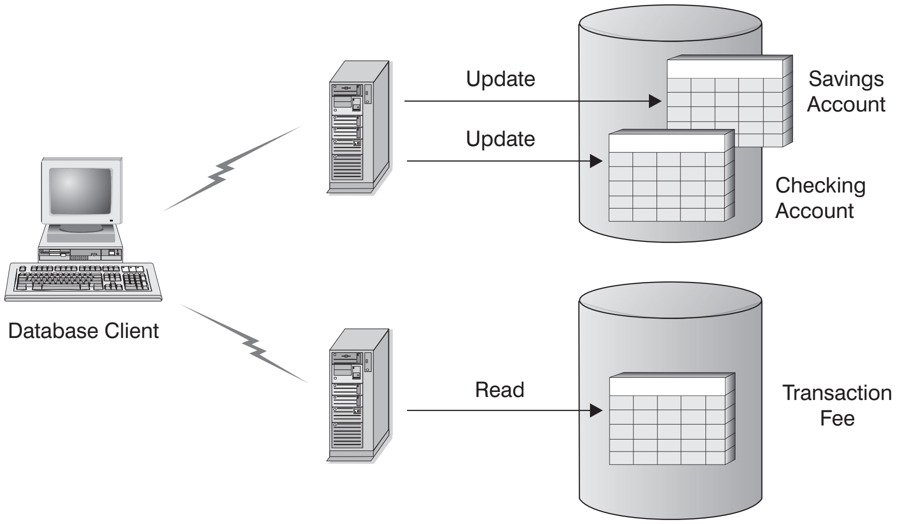


Figure 48. Using Multiple Databases in a Single Transaction

Figure 48 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.

Procedure:

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

1. Create the necessary tables in the appropriate databases
2. If physically remote, set up the database servers to use the appropriate communications protocols
3. If physically remote, catalog the nodes and the databases to identify the databases on the database servers
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).

If databases are located on a host or iSeries database server, you require DB2 Connect for connectivity to these servers.

Related concepts:

- "Units of work" on page 153

Related tasks:

- "Updating a single database in a transaction" on page 154
- "Updating multiple databases in a transaction" on page 156

Related reference:

- "PRECOMPILE Command" in the *Command Reference*

Updating multiple databases in a transaction

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*.

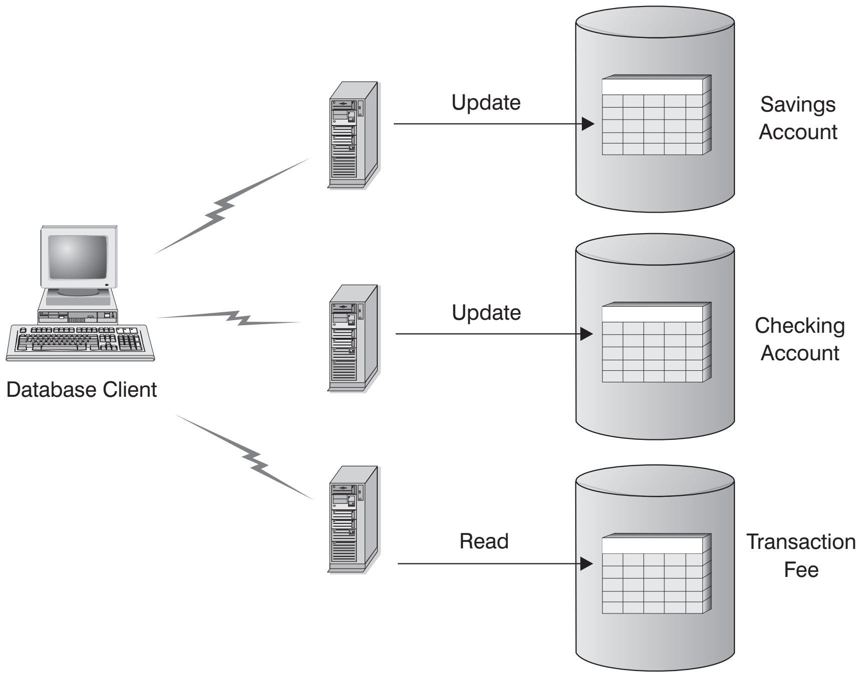


Figure 49. Updating Multiple Databases in a Single Transaction

Figure 49 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.

Procedure:

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

1. With the DB2 transaction manager (TM):
   a. Create the necessary tables in the appropriate databases
   b. If physically remote, set up the database servers to use the appropriate communications protocols
   c. If physically remote, catalog the nodes and the databases to identify the databases on the database servers
   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)
   e. Configure the DB2 transaction manager (TM).

2. Using an XA-compliant transaction manager:
   a. Create the necessary tables in the appropriate databases
   b. If physically remote, set up the database servers to use the appropriate communications protocols
   c. If physically remote, catalog the nodes and the databases to identify the databases on the database servers
   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)
   e. Configure the XA-compliant transaction manager to use the DB2 databases.

Related concepts:
- "Units of work" on page 153
- "DB2 transaction manager" on page 158

Related tasks:
- "Updating a single database in a transaction" on page 154
- "Updating a single database in a multi-database transaction" on page 155

Related reference:
- "PRECOMPILE Command" in the *Command Reference*

DB2 transaction manager

The DB2® transaction manager (TM) assigns identifiers to transactions, monitors their progress, and takes responsibility for transaction completion and failure. DB2 Universal Database™ (UDB) and DB2 Connect™ provide a transaction manager. The DB2 TM stores transaction information in the designated TM database.

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.

You can use the DB2 transaction manager with DB2 databases. If you have resources other than DB2 databases that you want to participate in a two-phase commit transaction, you can use an XA-compliant transaction manager.

Related concepts:

- "Units of work" on page 153
- "DB2 transaction manager configuration" on page 159
- "Two-phase commit" on page 165

DB2 transaction manager configuration

If you are using an XA-compliant transaction manager, such as IBM® WebSphere, BEA Tuxedo, or Microsoft® Transaction Server, you should follow the configuration instructions for that product.

When using DB2® UDB for UNIX-based systems or the Windows® operating system to coordinate your transactions, you must fulfill certain configuration requirements. If you use TCP/IP exclusively for communications, and DB2 UDB for Unix, Windows, iSeries™ V5, z/OS™ or OS/390® are the only database servers involved in your transactions, configuration is straightforward.

DB2 for Unix and Windows and DB2 for z/OS, OS/390, and iSeries V5 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 z/OS, OS/390, and iSeries V5) use TCP/IP exclusively.
- DB2 UDB for UNIX® based systems, Windows operating systems, z/OS, OS/390 or iSeries V5 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 iSeries database servers for multisite updates.

    Note: You should consider switching to TCP/IP as SNA may no longer be supported in future releases of DB2. SNA requires significant configuration knowledge and the configuration process itself can prove to be error prone. TCP/IP is simple to configure, has lower maintenance costs, and provides superior performance.
  - An XA-compliant transaction manager (such as IBM WebSphere) is coordinating the two-phase commit.

    This applies to both SNA and TCP/IP connectivity with host or iSeries database servers. 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 the database manager.

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*. Consider the following factors when setting this configuration parameter:

- The transaction manager database can be:
  - A DB2 UDB for UNIX or Windows Version 8 database
  - A DB2 for z/OS and OS/390 Version 7 database or a DB2 for OS/390 Version 5 or 6 database
  - A DB2 for iSeries V5 database

    DB2 for z/OS, OS/390, and iSeries V5 are the recommended database servers to use as the transaction manager database. z/OS, OS/390, and iSeries V5 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.

  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 AS/400® V4, or DB2 for VM&VSE is accessed
- DB2 for z/OS, OS/390, or iSeries V5 is accessed using SNA
- The DB2 Connect sync point manager is used to access host or iSeries 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*. Consider the following factors when setting this configuration parameter when using a Version 8 DB2 Run-Time Client:

- The transaction manager database can be a DB2 UDB for UNIX, Windows, z/OS, OS/390, or iSeries V5 database. The transaction manager database cannot be a DB2 UDB for Unix, Windows, or OS/2® Version 7 database.
- The user ID and password used will be those of the first database accessed.
- 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.
  - The first database connected is an acceptable database to be used as the transaction manager.

  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.

Configuration parameters

You should consider the following configuration parameters when you are setting up your environment.

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.
- *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.

Related concepts:

- "DB2 transaction manager" on page 158

Related tasks:

- "Configuring IBM TXSeries CICS" on page 191
- "Configuring IBM TXSeries Encina" on page 191
- "Configuring BEA Tuxedo" on page 193
- "Configuring IBM WebSphere Application Server" on page 191

Related reference:

- "Sync Point Manager Log File Path configuration parameter - spm_log_path" in the *Administration Guide: Performance*

- "Auto Restart Enable configuration parameter - autorestart" in the *Administration Guide: Performance*
- "Maximum Number of Active Applications configuration parameter - maxappls" in the *Administration Guide: Performance*
- "Transaction Resync Interval configuration parameter - resync_interval" in the *Administration Guide: Performance*
- "Transaction Manager Database Name configuration parameter - tm_database" in the *Administration Guide: Performance*
- "Sync Point Manager Name configuration parameter - spm_name" in the *Administration Guide: Performance*
- "Sync Point Manager Log File Size configuration parameter - spm_log_file_sz" in the *Administration Guide: Performance*
- "Sync Point Manager Resync Agent Limit configuration parameter - spm_max_resync" in the *Administration Guide: Performance*

Updating a database from a host or iSeries client

Applications executing on host or iSeries can access data residing on DB2 UDB for Unix and Windows database servers. Both TCP/IP and SNA can be used for this access. If SNA is used, the sync point manager (SPM) is required on the DB2 UDB server. If TCP/IP is used the SPM is not used.

Note: If you are using SNA, you should consider switching to TCP/IP as SNA may no longer be supported in future releases of DB2. SNA requires significant configuration knowledge and the configuration process itself can prove to be error prone. TCP/IP is simple to configure, has lower maintenance costs, and provides superior performance.

The database server that is being accessed from the host or the iSeries database client does not have to be local to the workstation with the DB2 sync point manager. The host or iSeries 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 iSeries database clients.

Procedure:

On the workstation that will be directly accessed by the host or iSeries application:

1. Install DB2 Universal Database Enterprise Server Edition, to provide multisite update support with the host or iSeries 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` and/or `TCP/IP`.
5. If SNA is used from the host or iSeries client, the sync point manager must be configured:
   a. 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.
   b. 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.
   c. If necessary, update *spm_name* on the DB2 Universal Database server, using the UPDATE DATABASE MANAGER CONFIGURATION command.
6. Configure communications required for this DB2 workstation to connect to remote DB2 UDB servers, if any.
7. Configure communications required for remote DB2 UDB servers to connect to this DB2 server.
8. Stop and restart the database manager on the DB2 Universal Database server.

   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 iSeries 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.

Related reference:
- "Sync Point Manager Name configuration parameter - spm_name" in the *Administration Guide: Performance*

- “Communications variables” in the *Administration Guide: Performance*

Two-phase commit

Figure 50 on page 166 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.

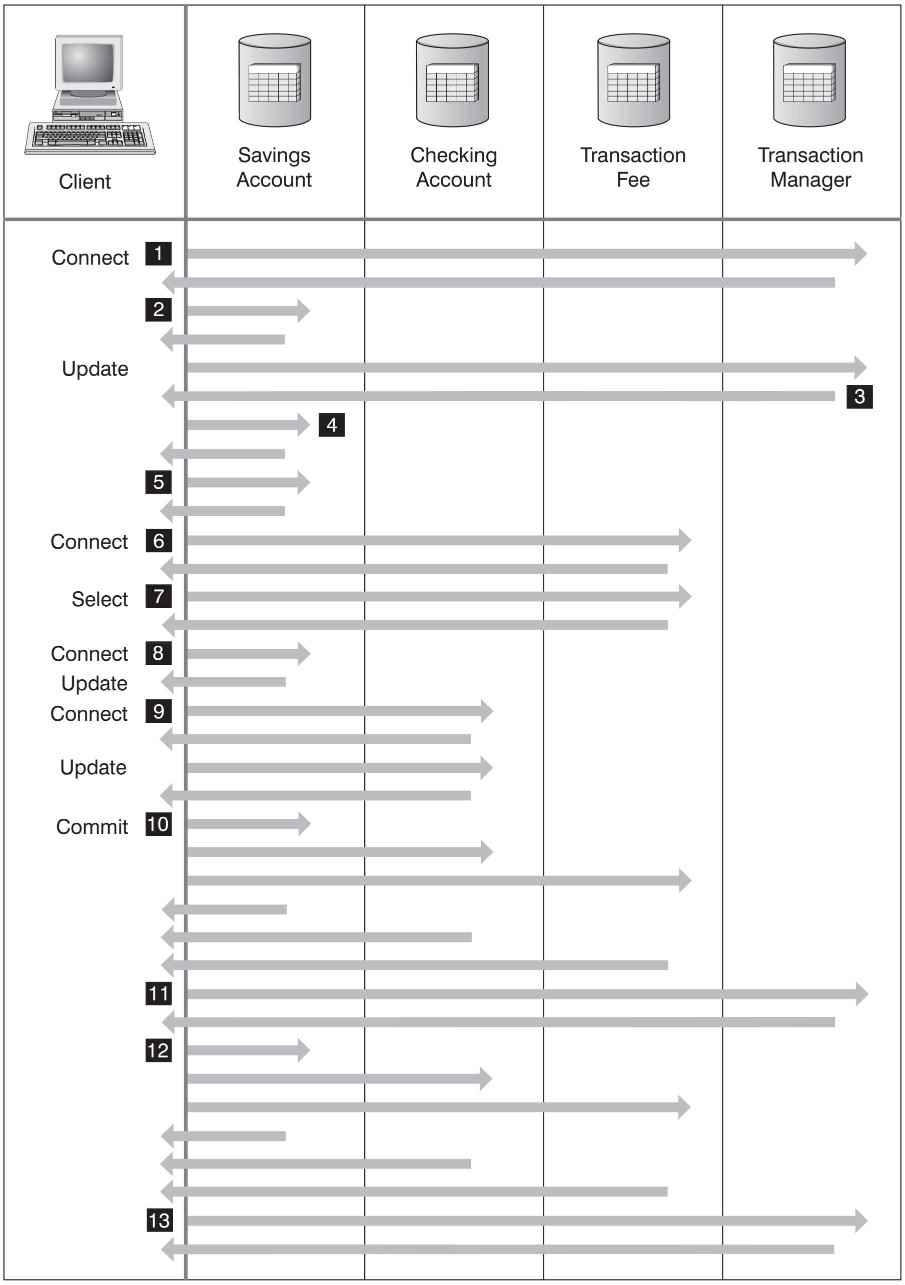


Figure 50. Updating Multiple Databases

0 The application is prepared for two-phase commit. This can be accomplished through precompilation options. This can also be accomplished through DB2® CLI (Call Level Interface) configuration.

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.

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.

Related concepts:

- "Units of work" on page 153
- "DB2 transaction manager" on page 158

Error recovery 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. 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* 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″.

Error recovery if autorestart=off

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.

Related concepts:

- "Two-phase commit" on page 165

Related tasks:

- "Manually resolving indoubt transactions" on page 182

Related reference:

- "Auto Restart Enable configuration parameter - autorestart" in the *Administration Guide: Performance*
- "LIST INDOUBT TRANSACTIONS Command" in the *Command Reference*
- "TERMINATE Command" in the *Command Reference*
- "RESTART DATABASE Command" in the *Command Reference*

Chapter 7. Designing for XA-compliant 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 in a transaction" on page 156.

The following topics will assist you in using the database manager with an XA-compliant transaction manager, such as IBM WebSphere or BEA Tuxedo.

- "X/Open distributed transaction processing model" on page 172
- "Resource manager setup" on page 176
- "xa_open string formats" on page 176
- "Manually resolving indoubt transactions" on page 182
- "Security considerations for XA transaction managers" on page 185
- "Configuration considerations for XA transaction managers" on page 186
- "XA function supported by DB2 UDB" on page 188
- "XA interface problem determination" on page 190
- "Configuring IBM WebSphere Application Server" on page 191
- "Configuring IBM TXSeries CICS" on page 191
- "Configuring IBM TXSeries Encina" on page 191
- "Configuring BEA Tuxedo" on page 193

If you are looking for information about Microsoft Transaction Server, see the *CLI Guide and Reference, Volume 1*.

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/udb/winos2unix/support`

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)
- Resources managers (RM)

Figure 51 illustrates this model, and shows the relationship among these components.

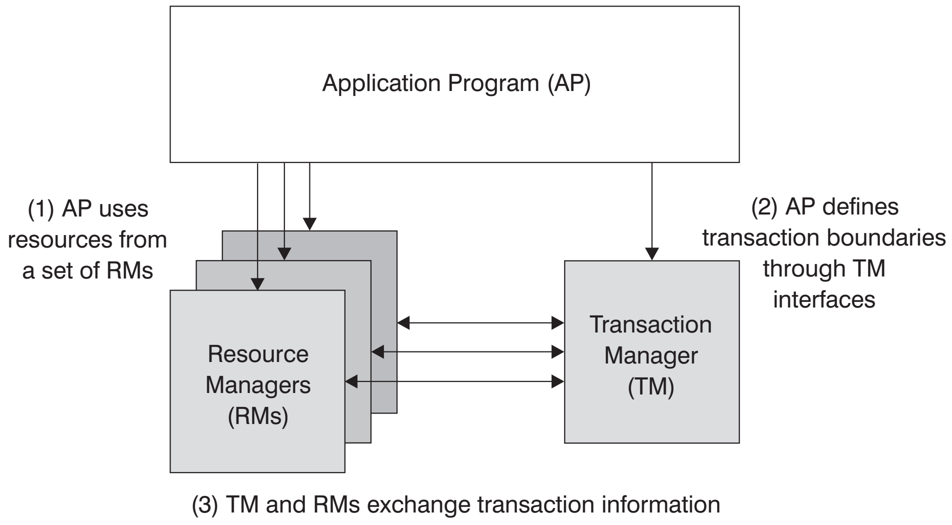

Application Program (AP)
(1) AP uses resources from a set of RMs
(2) AP defines transaction boundaries through TM interfaces
Resource Managers (RMs)
Transaction Manager (TM)
(3) TM and RMs exchange transaction information

Figure 51. 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.

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.

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.

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.

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.

Related concepts:

- "Security considerations for XA transaction managers" on page 185
- "XA function supported by DB2 UDB" on page 188
- "X/Open XA Interface Programming Considerations" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- “Updating multiple databases in a transaction” on page 156

Resource manager setup

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.

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.

Database connection considerations

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.

Related concepts:

- “X/Open distributed transaction processing model” on page 172

Related reference:

- “xa_open string formats” on page 176

xa_open string formats

xa_open string format for DB2 Version 7 and later

This is the format for the xa_open string:

```
   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 21. 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 the next table. 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.				
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 eliminating 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.				

Table 21. Valid Values for parm_id (continued)

Parameter Name	Value	Mandatory?	Case Sensitive?	Default 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. If SUSPEND_CURSOR is off, all cursors except cursors with hold attributes are closed. On resumption of the suspended transaction, the application must obtain the cursors again. If SUSPEND_CURSOR is on, any open cursors are not closed, and are available to the suspended transaction on resumption. 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.				
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_CURSORS is on, cursors with hold attributes are not closed, and will persist across transaction commit boundaries. When using this option, the global transaction must be committed or rolled back from the same thread of control. If HOLD_CURSOR is off, the opening of any cursors with hold attributes will be rejected. 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 22. 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`

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.

Table 22. Valid Values for TPM and tp_mon_name (continued)

TPM Value	TP Monitor Product	Internal Settings
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. Therefore this TPM value is ignored.

xa_open string format for earlier versions

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 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.

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
  ```

Related concepts:
- "X/Open distributed transaction processing model" on page 172

Related reference:
- "Transaction Processor Monitor Name configuration parameter - tp_mon_name" in the *Administration Guide: Performance*

Updating host or iSeries database servers with an XA-compliant transaction manager

Host and iSeries database servers may be updatable depending upon the architecture of the XA Transaction Manager.

Procedure:

To support commit sequences from different processes, the DB2 Connect connection concentrator must be enabled. To enable the DB2 Connect connection concentrator, set the database manager configuration parameter *max_connections* to a value greater then *maxagents*. Note that the DB2 Connect connection concentrator requires a DB2 Version 7.1 client or later to support XA commit sequences from different processes.

You will also require DB2 Connect with the DB2 sync point manager (SPM) configured.

Related reference:
- "Maximum Number of Agents configuration parameter - maxagents" in the *Administration Guide: Performance*
- "Maximum Number of Client Connections configuration parameter - max_connections" in the *Administration Guide: Performance*

Manually resolving indoubt transactions

An XA-compliant transaction manager (Transaction Processing Monitor) uses a two-phase commit process similar to that used by the DB2 transaction manager. 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 that occur for the DB2 transaction manager 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.

Restrictions:

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. Moreover, an offline backup cannot be taken unless all indoubt transactions have been resolved.

The heuristic forget function is required in the following situations:

- when a heuristically committed or rolled back transaction causes a log full condition, indicated in output from the LIST INDOUBT TRANSACTIONS command

- when an offline backup is to be taken

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.

Procedure:

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.

To obtain indoubt transaction information from DB2 UDB on UNIX or Windows, connect to the database and issue the LIST INDOUBT TRANSACTIONS WITH PROMPTING command, or the equivalent API.

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

- You can obtain indoubt information directly from the host or iSeries server.

  To obtain indoubt information directly from DB2 for z/OS and 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 iSeries, invoke the **wrkcmtdfn** command.
- You can obtain indoubt information from the DB2 Connect server used to access the host or iSeries 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.

Related concepts:

- “Two-phase commit” on page 165

Related reference:

- “LIST INDOUBT TRANSACTIONS Command” in the *Command Reference*
- “LIST DRDA INDOUBT TRANSACTIONS Command” in the *Command Reference*

Security considerations for XA transaction managers

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.

Related concepts:
- “X/Open distributed transaction processing model” on page 172

Configuration considerations for XA transaction managers

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.

- *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 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.

Related concepts:

- "X/Open distributed transaction processing model" on page 172

Related reference:

- "Auto Restart Enable configuration parameter - autorestart" in the *Administration Guide: Performance*

- "APPC Transaction Program Name configuration parameter - tpname" in the *Administration Guide: Performance*
- "Maximum Number of Active Applications configuration parameter - maxappls" in the *Administration Guide: Performance*
- "Transaction Manager Database Name configuration parameter - tm_database" in the *Administration Guide: Performance*
- "Transaction Processor Monitor Name configuration parameter - tp_mon_name" in the *Administration Guide: Performance*
- "LIST INDOUBT TRANSACTIONS Command" in the *Command Reference*
- "RESTART DATABASE Command" in the *Command Reference*

XA function supported by DB2 UDB

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.
- Association migration

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

XA switch usage and location

As required by the XA interface, the database manager provides a *db2xa_switch* external C variable of type xa_switch_t to return the XA switch structure to the TM. Other than the addresses of 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

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`.

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 ;
   }
```

Related concepts:

- "X/Open distributed transaction processing model" on page 172

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.

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.

Related concepts:
- “X/Open distributed transaction processing model” on page 172

Related reference:
- “xa_open string formats” on page 176

XA transaction manager configuration

Configuring IBM WebSphere Application Server

IBM WebSphere Application Server is a Java-based application server. It can use DB2’s XA support via the Java Transaction API (JTA) provided by the DB2 JDBC driver. Refer to IBM WebSphere documentation regarding how to use the Java Transaction API with WebSphere Application Server. WebSphere Application Server documentation can be viewed online at http://www-4.ibm.com/software/webservers/appserv/infocenter.html.

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 iSeries 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 z/OS and OS/390, DB2 for iSeries, 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 iSeries 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 Configuration Assistant, 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.

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.

If you are accessing DB2 for z/OS and OS/390, DB2 for iSeries, or DB2 for VSE & VM, you must use the DB2 Syncpoint Manager.

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)
   ```

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 `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.

Related concepts:

- “DB2 Connect and transaction processing monitors” in the *DB2 Connect User’s Guide*

Related reference:

- “Transaction Processor Monitor Name configuration parameter - tp_mon_name” in the *Administration Guide: Performance*
- “xa_open string formats” on page 176

Configuring BEA Tuxedo

Procedure:

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 z/OS and OS/390, DB2 for iSeries, or DB2 for VM&VSE, the DB2 Connect XA concentrator will be required.
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.

Related concepts:

- "DB2 Connect and transaction processing monitors" in the *DB2 Connect User's Guide*

Related reference:

- "Transaction Processor Monitor Name configuration parameter - tp_mon_name" in the *Administration Guide: Performance*
- "LIST APPLICATIONS Command" in the *Command Reference*

Part 3. Appendixes

Appendix A. 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 7 and Version 8 are described. They are grouped according to the following categories:

- System Catalog Information
- Application Programming
- SQL
- Database Security and Tuning
- Utilities and Tools
- Connectivity and Coexistence
- Messages
- 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 (for Version 7 only)

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.

System catalog information

PK_COLNAMES and FK_COLNAMES in a future version of DB2 Universal Database

WIN	UNIX

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

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.

Utilities and tools

Support for re-creation of type-1 indexes will be removed

WIN	UNIX

Change: A new type of index is introduced in Version 8, called a type-2 index. In type-1 indexes, that is indexes created prior to Version 8, a key is physically removed from a leaf page as part of the deletion or update of a table row. In type-2 indexes, keys are marked as deleted when a row is deleted or updated, but they are not physically removed until after the deletion or update has committed. When support for re-creation of type-1 indexes is removed, you will not have to rebuild your indexes manually. Type-1 indexes will continue to function correctly. All actions that result in the re-creation of indexes will automatically convert type-1 indexes to type-2 indexes. In a future version, support for type-1 indexes will be removed.

Explanation: Type-2 indexes have advantages over type-1 indexes:

- a type-2 index can be created on columns whose length is greater than 255 bytes
- the use of next-key locking is reduced to a minimum, which improves concurrency.

Resolution: Develop a plan to convert your existing indexes to type-2 indexes over time. The Online Index Reorganization capability can help do this while minimizing availability outages. Increase index table space size if needed. Consider creating new indexes in large table spaces and moving existing indexes to large table spaces.

Version 8 incompatibilities between releases

System Catalog Information

IMPLEMENTED column in catalog tables

WIN	UNIX

Change: In previous versions, the column IMPLEMENTED in SYSIBM.SYSFUNCTIONS and SYSCAT.SYSFUNCTIONS had values of Y, M, H, and N. In Version 8, the values are Y and N.

Resolution: Recode your applications to use only the values Y and N.

OBJCAT views renamed to SYSCAT views

WIN	UNIX

Change: The following OBJCAT views have been renamed to SYSCAT views: TRANSFORMS, INDEXEXTENSIONS, INDEXEXTENSIONMETHODS, INDEXEXTENSIONDEP, INDEXEXTENSIONPARMS, PREDICATESPECS, INDEXEXPLOITRULES.

Resolution: Recode your applications to use the SYSCAT views.

SYSCAT views are now read-only

WIN	UNIX

Change: As of Version 8, the SYSCAT views are read-only.

Symptom: An UPDATE or INSERT operation on a view in the SYSCAT schema now fails.

Explanation: The SYSSTAT views are the recommended way to update the system catalog information. Some SYSCAT views were unintentionally updatable and this has now been fixed.

Resolution: Change your applications to reference the updatable SYSSTAT views instead.

Application programming

SQL0818N error not returned when using VERSION option

WIN	UNIX

Change: If you use the new VERSION option on the PRECOMPILE, BIND, REBIND, and DROP PACKAGE commands, requests to execute may now return an SQL0805N error instead of an SQL0818N error.

Symptom: Applications coded to react to an SQL0818N error may not behave as before.

Resolution: Recode your applications to react to both SQL0805N and SQL0818N errors.

SQL0306N error not returned to the precomipler when a host variable is not defined

WIN	UNIX

Change: If a host variable is not declared in the BEGIN DECLARE section and is used in the EXEC SQL section, SQL0306N will not be returned by the precompiler. If the variable is declared elsewhere in the application, application runtime will return SQL0804N. If the variable is not declared anywhere in the application, the compiler will return an error at compilation time.

Symptom: Applications coded to react to an SQL0306N error at precompilation time may not behave as before.

Resolution: Host variables should be declared in the BEGIN DECLARE section. If host variables are declared in a section other than the BEGIN DECLARE section, you should recode your application to handle SQL0804 return codes.

Data types not supported for use with scrollable cursors

WIN	UNIX

Change: Scrollable cursors using LONG VARCHAR, LONG VARGRAPHIC, DATALINK and LOB types, distinct types on any of these types, or structured types will not be supported in Version 8. Any of these data types supported for Version 7 scrollable cursors will no longer be supported.

Symptom: If any columns with these data types are specified in the select list of a scrollable cursor, SQL0270N Reason Code 53 is returned.

Resolution: Modify the select-list of the scrollable cursor so it does not include a column with any of these types.

Euro version of code page conversion tables

WIN	UNIX

Change: The Version 8 code page conversion tables, which provide support for the euro symbol, are slightly different from the conversion tables supplied with previous versions of DB2.

Resolution: If you want to use the pre-Version 8 code page conversion tables, they are provided in the directory `sqllib/conv/v7`.

Switching between a LOB locator and a LOB value

WIN	UNIX

Change: The ability to switch between a large object (LOB) locator and a LOB value has been changed during bindout on a cursor statement. When an application is bound with SQLRULES DB2 (the default behavior), the user will not be able to switch between LOB locators and LOB values.

Resolution: If you want to switch between a LOB locator and a LOB value during bindout of a cursor statement, precompile your application with SQLRULES STD.

Uncommitted units of work on UNIX platforms

	UNIX

Change: Prior to version 8, UNIX-based applications that did not use either explicit or implicit context support would commit an outstanding unit of work if the application terminated normally without directly invoking either a CONNECT RESET, COMMIT, OR ROLLBACK statement. CLI, ODBC, and Java-based applications (implicit context support) and applications that would explictly create application contexts would always roll back any oustanding unit of work if the application terminated. Abnormal application termination would also lead to an implict ROLLBACK for the outstanding unit of work. In Version 8 all application terminations will implicitly roll back the outstanding unit of work. Windows-based applications will not change as they already perform an implicit ROLLBACK for normal or abnormal application termination.

Resolution: In order to ensure that transactions are committed, the application should perform either an explicit COMMIT or a CONNECT RESET before terminating.

Change to savepoint naming

WIN	UNIX

Change: Savepoint names can no longer start with ″SYS″.

Symptom: Creating a savepoint with a name that starts with ″SYS″ will fail with error SQL0707N.

Explanation: Savepoint names that start with ″SYS″ are reserved for use by the system.

Resolution: Rename any savepoints that start with ″SYS″ to another name that does not start with ″SYS″.

Code page conversion errors and byte substitution

WIN	UNIX

Change: Code page conversion, when necessary, will now be performed during the bind in phase.

Symptom: Code page conversion errors and byte substitution can happen in new situations. For example, the following statements will now convert the data in the host variable :hv from the application code page to the database code page, if these code pages are not the same:

```
SELECT :hv FROM table
VALUES :hv
```

Explanation: The data supplied could not be converted to the database code page without error or byte substitution. Previously, code page conversion did not take place, now code page conversion is performed.

Resolution: Change the data, the application code page or the database code page so that code page conversion does not produce errors or byte substitution, or code the application to handle the possibility of code page conversion errors or byte substitution.

Code page conversion for host variables

WIN	UNIX

Change: Code page conversion, when necessary, will now be performed during the bind in phase.

Symptom: Different results.

Explanation: Now that code page conversion, when necessary, will always be done for host variables, predicate evaluation will always occur in the database code page and not the application code page. For example,

```
SELECT * FROM table WHERE :hv1 > :hv2
```

will be done using the database code page rather than the application code page. The collation used continues to be the database collation.

Resolution: Verify that the results in previous versions were indeed the desired results. If they were, then change the predicate to produce the desired result given that the database collation and code page are used. Alternatively, change the application code page or the database code page.

Expansion and contraction of data in host variables

WIN	UNIX

Change: Code page conversion, when necessary, will now be performed during the bind in phase.

Symptom: Data from host variables have a different length.

Explanation: Since expansion or contraction can occur during code page conversion, operations that depend on the length of the data in the host variable can produce different results or an error situation.

Resolution: Change the data, the application code page or the database code page so that code page conversion does not produce changes in length of the converted data, or code the application to handle the possibility of code page conversion causing the length of the data to change.

Length of host variables after code page conversion

WIN	UNIX

Change: Code page conversion will no longer cause result length to increase for host variables or parameter markers due to expansion.

Symptom: Data truncation errors.

Explanation: The length of the character data type determined for the untyped parameter marker is no longer increased to account for potential expansion from code page conversion. The result length will be shorter for operations that determine result length using the length of the untyped parameter marker. For example, given that C1 is a CHAR(10) column:

```
   VALUES CONCAT (?, C1)
```

no longer has a result data type and length of CHAR(40) for a database where 3 times expansion is possible when converting from the application code page to the database code page, but will have a result data type and length of CHAR(20).

Resolution: Use a CAST to give the untyped parameter marker the type desired or change the operand that determines the type of the untyped parameter marker to a data type or length that would accommodate the expansion of the data due to code page conversion.

Change to output of DESCRIBE statement

WIN	UNIX

Change: Code page conversion will no longer cause result length to increase for host variables or parameter markers due to expansion.

Symptom: Output from DESCRIBE statement changes.

Explanation: Since the result length is not increased due to potential expansion on code page conversion, the output of a DESCRIBE statement that describes such a result length will now be different.

Resolution: If necessary, change the application to handle the new values returned from the DESCRIBE statement.

Error when using SUBSTR function with host variables

WIN	UNIX

Change: Code page conversion will no longer cause result length to increase for host variables or parameter markers due to expansion.

Symptom: Error SQL0138N from SUBSTR.

Explanation: Potential expansion due to code page conversion was taken into account by increasing the length set aside for the host variable. This allowed, for example, `SUBSTR (:hv,19,1)` to work successfully for a host variable with a length of 10. This will no longer work.

Resolution: Increase the length of the host variable to account for the length of the converted data or change the SUBSTR invocation to specify positions within the length of the host variable.

Non-thread safe libraries are no longer supported on Solaris

	UNIX

Change: The non-thread safe library `libdb2_noth.so` is no longer available.

Symptom: Tools or applications that require `libdb2_noth.so` will not work.

Explanation: Since support for the obsolete non-thread safe libraries is no longer required, the `libdb2_noth.so` library is not included with DB2 UDB for Solaris.

Resolution: Change the tool or application to use the thread-safe `libdb2.so` library instead. Re-link your applications with the -mt parameter.

SQL

Identical specific names not permitted for functions and procedures

WIN	UNIX

Change: The name space for SPECIFICNAME has been unified. Previous versions of DB2 would allow a function and a procedure to have the same specific name, but Version 8 does not allow this.

Symptom: If you are migrating a database to Version 8, the db2ckmig utility will check for functions and procedures with the same specific name. If duplicate names are encountered during migration, the migration will fail.

Resolution: Drop the procedure and recreate it with a different specific name.

EXECUTE privilege on functions and procedures

WIN	UNIX

Change: Previously, a user only had to create a routine for others to be able to use it. Now after creating a routine, a user has to GRANT EXECUTE on it first before others can use it.

In previous versions, there were no authorization checks on procedures, but the invoker had to have EXECUTE privilege on any package invoked from the procedure. For an embedded application precompiled with CALL_RESOLUTION IMMEDIATE in Version 8, and for a CLI cataloged procedure, the invoker has to have EXECUTE privilege on the procedure and only the definer of the procedure has to have EXECUTE privilege on any packages.

Symptom:

1. An application may not work correctly.
2. An existing procedure that is made up of multiple packages, and for which the definer of the procedure does not have access to all the packages, will not work correctly.

Resolution:

1. Issue the required GRANT EXECUTE statements. If all the routines are in a single schema, the privileges for each type of routine can be granted with a single statement, for example:
   ```
   GRANT EXECUTE ON FUNCTION schema1.* TO PUBLIC
   ```
2. If one package is usable by everyone but another package is restricted to a few privileged users, a stored procedure that uses both packages will watch for an authority error when it tries to access the second package. If it sees the authority error, it knows that the user is not a privileged user and the procedure bypasses part of its logic.

   You can resolve this in several ways:

a. When precompiling a program, CALL_RESOLUTION DEFERRED should be set to indicate that the program will be executed as an invocation of the deprecated sqleproc() API when the precompiler fails to resolve a procedure on a CALL statement.
b. The CLI keyword UseOldStpCall can be added to the db2cli.ini file to control the way in which procedures are invoked. It can have two values: A value of *0* means procedures will not be invoked using the old call method, while a value of *1* means procedures will be invoked using the old call method.
c. Grant EXECUTE privilege to everyone who executes the package.

Adding a foreign key constraint to a table

WIN	UNIX

Change: In previous versions, if you created a foreign key constraint that referenced a table in check pending state, the dependent table would also be put into check pending state. In Version 8, if you create a foreign key constraint that references a table in check pending state, there are two possible results:

1. If the foreign key constraint is added upon creation of the dependent table, the creation of the table and the addition of the constraint will be successful because the table will be created empty, and therefore no rows will violate the constraint.
2. If a foreign key is added to an existing table, you will receive error SQL0668N.

Resolution: Use the SET INTEGRITY ... IMMEDIATE CHECKED statement to turn on integrity checking for the table that is in check pending state, before adding the foreign key that references the table.

Change to SET INTEGRITY ... IMMEDIATE CHECKED

WIN	UNIX

Change: In previous releases, a table that had the SET INTEGRITY ... UNCHECKED statement issued on it (i.e. with some 'U' bytes in the const_checked column of SYSCAT.TABLES) would by default be fully processed upon the next SET INTEGRITY ... IMMEDIATE CHECKED statement, meaning all records would be checked for constraint violations. You had to explicitly specify INCREMENTAL to avoid full processing.

In Version 8, when the SET INTEGRITY ... IMMEDIATE CHECKED statement is issued, the default is to leave the unchecked data alone (i.e. keeping the 'U' bytes) by doing only incremental processing. (A warning will be returned that old data remains unverified.)

Explanation: This change is made to avoid having the default behavior be a constraint check of all records, which usually consumes more resources.

Resolution: You will have to explicitly specify NOT INCREMENTAL to force full processing.

Decimal separator for CHAR function

WIN	UNIX

Change: Dynamic applications that run on servers with a locale that uses the comma as the decimal separator and include unqualified invocations of the CHAR function with an argument of type REAL or DOUBLE, will return a period as the separator character in the result of the CHAR(double) function. This incompatibility will also be visible when objects like views and triggers are re-created in Version 8 or when static packages are explicitly rebound.

Explanation: This is a result of resolving to the new SYSIBM.CHAR(double) function signature instead of the SYSFUN.CHAR(double) signature.

Resolution: To maintain the behavior from earlier versions of DB2, the application will need to explicitly invoke the function with SYSFUN.CHAR instead of allowing function resolution to select the SYSIBM.CHAR signature.

Changes to CALL statement

WIN	UNIX

Change: In Version 8, an application precompiled with CALL_RESOLUTION IMMEDIATE and a CLI cataloged procedure have several key differences compared to previous versions:

- Host variable support has been replaced by support for dynamic CALL.
- Support for compilation of applications that call uncataloged stored procedures has been removed. Uncataloged stored procedure support will be removed entirely in a future version of DB2.
- Variable argument list stored procedure support has been deprecated.
- There are different rules for loading the stored procedure library.

Resolution: The CALL statement as supported prior to Version 8 will continue to be available and can be accessed using the CALL_RESOLUTION DEFERRED option on the PRECOMPILE PROGRAM command.

Existing applications (built prior to Version 8) will continue to work. If applications are re-precompiled without the CALL_RESOLUTION DEFERRED option, then source code changes may be necessary.

Support for the CALL_RESOLUTION DEFERRED statement will be removed in a future version.

Output from UDFs returning fixed-length strings

WIN	UNIX

Change: A UDF (scalar or table function) can be defined to return a fixed-length string (CHAR(n) or GRAPHIC(n)). In previous versions, if the returned value contains an imbedded null character, the result would simply be n bytes (or 2n bytes for GRAPHIC data types) including the null character and any bytes to the right of the null character. In Version 8, DB2 looks for the null character and returns blanks from that point (the null character) to the end of the value.

Resolution: If you want to continue the pre-Version 8 behavior, change the definition of the returned value from CHAR(n) to CHAR(n) FOR BIT DATA. There is no method to continue the pre-Version 8 behavior for GRAPHIC data.

Change in database connection behavior

WIN	UNIX

Change: In Version 7, if you use embedded SQL to connect to a database, and then attempt a connection to a non-existent database, the attempt to connect to the non-existent database will fail with SQL1013N. The connection to the first database still exists. In Version 8, the attempt to connect to the non-existent database will result in a disconnection from the first database. This will result in the application being left with no connection.

Resolution: Code your embedded SQL to reconnect to the initial database following an unsuccessful attempt to connect to another database.

Revoking CONTROL on packages

WIN	UNIX

Change: A user can grant privileges on a package using the CONTROL privilege. In DB2 Version 8, the WITH GRANT OPTION provides a mechanism to determine a user's authorization to grant privileges on packages to other users. This mechanism is used in place of CONTROL to determine whether a user may grant privileges to others. When CONTROL is revoked, users will continue to be able to grant privileges to others.

Symptom: A user can still grant privileges on a package, following the revocation of CONTROL privilege.

Resolution: If a user should no longer be authorized to grant privileges on packages to others, revoke all privileges on the package and grant only those required.

Error when casting a FOR BIT DATA character string to a CLOB

WIN	UNIX

Change: Casting a character string defined as FOR BIT DATA to a CLOB (using the CAST specification or the CLOB function) now returns an error (SQLSTATE 42846).

Symptom: Casting to a CLOB now returns an error where previously it did not.

Explanation: FOR BIT DATA is not supported for the CLOB data type. The result of using the CAST specification or the CLOB function when a FOR BIT DATA string is given as an argument is not defined. This situation in now caught as an error.

Resolution: Change the argument to the CAST specification or the CLOB function so that it is not a FOR BIT DATA string. This can be done by using the CAST specification to cast the FOR BIT DATA string to a FOR SBCS DATA string or a FOR MIXED DATA string. For example, if C1FBD is a VARCHAR(20) column declared as FOR BIT DATA, in a non-DBCS database, the following would be a valid argument to the CLOB function:

```
CAST (C1FBD AS VARCHAR(20) FOR SBCS DATA)
```

Database security and tuning

Authority for CREATE FUNCTION, CREATE METHOD and CREATE PROCEDURE statements

WIN	UNIX

Change: The CREATE_EXTERNAL_ROUTINE authority is introduced in Version 8.

Symptom: CREATE FUNCTION, CREATE METHOD and CREATE PROCEDURE statements with the EXTERNAL option may fail.

Resolution: Grant CREATE_EXTERNAL_ROUTINE authority to users who issue CREATE FUNCTION, CREATE METHOD and CREATE PROCEDURE statements with the EXTERNAL option.

Utilities and tools

Downlevel CREATE DATABASE and DROP DATABASE not supported

WIN	UNIX

Change: In Version 8, the CREATE DATABASE and DROP DATABASE commands are not supported from downlevel clients or to downlevel servers.

Symptom: You will receive error SQL0901N when you issue one of these commands.

Explanation: The CREATE DATABASE and DROP DATABASE commands are both only supported from Version 8 clients to Version 8 servers. You cannot issue these commands from a Version 6 or Version 7 client to a Version 8 server. You cannot issue these commands from a Version 8 client to a Version 7 server.

Resolution: Create or drop a Version 8 database from a Version 8 client. Create or drop a Version 7 database from a Version 6 or Version 7 client.

Mode change to tables after a load

WIN	UNIX

Change: In previous versions, a table that has been loaded with the INSERT option and has immediate materialized query tables (also known as summary tables) would be in Normal (Full Access) state after a subsequent SET INTEGRITY IMMEDIATE CHECKED statement on it. In Version 8, the table will be in No Data Movement mode after the SET INTEGRITY IMMEDIATE CHECKED statement.

Explanation: Access to a table in No Data Movement mode is very similar to a table in Normal (Full Access) mode, except for some statements and utilities that involve data movement within the table itself.

Resolution: You can force the base table that has been loaded and has dependent immediate summary tables to bypass the No Data Movement mode and to go directly into Full Access mode by issuing a SET INTEGRITY ... IMMEDIATE CHECKED FULL ACCESS statement on the base table. However, use of this option is not recommended as it will force a full refresh of the dependent immediate materialized query tables (also known as summary tables).

Load utility in insert or replace mode

WIN	UNIX

Change: In previous versions, when using the load utility in insert or replace mode, the default option was CASCADE IMMEDIATE when integrity checking was turned off; when the table was put into check pending state, all of its dependent foreign key tables and dependent materialized query tables (also known as summary tables) were also immediately put into check pending state.

For Version 8, when using the load utility in insert or replace mode, the default is CASCADE DEFERRED when integrity checking has been turned off.

Resolution: You can put dependent foreign key tables and dependent materialized query tables into check pending state along with their parent tables by using the CHECK PENDING CASCADE IMMEDIATE option of the LOAD command.

Connectivity and coexistence

Down level server support

WIN	UNIX

Change: As you move your environment from Version 7 to Version 8, if you are in a situation where you migrate your client machines to Version 8 before you migrate all of your servers to Version 8, there are several restrictions and limitations. These restrictions and limitations are not associated with DB2 Connect; nor with zSeries, OS/390, or iSeries database servers.

Resolution: For Version 8 clients to work with Version 7 servers, you need to configure/enable the use of DRDA Application Server capability on the Version 7 server. For information on how to do this, refer to the Version 7 *Installation and Configuration Supplement*.

To avoid the known restrictions and limitations, you should migrate all of your servers to Version 8 before you migrate any of your client machines to

Version 8. If this is not possible, then you should know that when accessing Version 7 servers from Version 8 clients, there is no support available for:

- Some data types:
  - Large object (LOB) data types.
  - User-defined distinct types (UDTs).
  - DATALINK data types.

    The DATALINK data type allows for the management of external data found in non-relational storage. The DATALINK data type, references files that physically reside on file systems external to DB2 Universal Database.
- Some security capabilities:
  - Authentication type SERVER_ENCRYPT.

    SERVER_ENCRYPT is a method to encrypt a password. The encrypted password is used with the user ID to authenticate the user.
  - Changing passwords.

    You are not able to change passwords on the Version 7 server from a Version 8 client.
- Certain connections and communication protocols:
  - Instance requests that require an ATTACH instead of a connection.

    ATTACH is not supported from a Version 8 client to a Version 7 server.
  - The only supported network protocol is TCP/IP.

    Other network protocols like SNA, NetBIOS, IPX/SPX, and others are not supported.
- Some application features and tasks:
  - The DESCRIBE INPUT statement is not supported with one exception for ODBC/JDBC applications.

    In order to support Version 8 clients running ODBC/JDBC applications accessing Version 7 servers, a fix for DESCRIBE INPUT support must be applied to all Version 7 servers where this type of access is required. This fix is associated with APAR IY30655 and will be available before the Version 8 General Availability date. Use the "Contacting IBM" information in any DB2 Universal Database document to find out how to get the fix associated with APAR IY30655.

    The DESCRIBE INPUT statement is a performance and usability enhancement to allow an application requestor to obtain a description of input parameter markers in a prepared statement. For a CALL statement, this includes the parameter markers associated with the IN and INOUT parameters for the stored procedure.
  - Two-phase commit.

    The Version 7 server cannot be used as a transaction manager database when using coordinated transactions that involve Version 8 clients. Nor

  can a Version 7 server participate in a coordinated transaction where a Version 8 server may be the transaction manager database.
  - XA-compliant transaction managers.

    An application using a Version 8 client cannot use a Version 7 server as an XA resource. This includes WebSphere, Microsoft COM+/MTS, BEA WebLogic, and others that are part of a transaction management arrangement.
  - Monitoring.
  - Utilities.

    Those utilities that can be initiated by a client to a server are not supported when the client is at Version 8 and the server is at Version 7.
  - SQL statements greater than 32 KB in size.

In addition to these limitations and restrictions for Version 8 clients working with Version 7 servers, there are also similar limitations and restrictions for Version 8 tools working with Version 7 servers.

The following Version 8 tools support only Version 8 servers:
- Control Center
- Task Center
- Journal
- Satellite Administration Center
- Information Catalog Center (including the Web-version of this center)
- Health Center (including the Web-version of this center)
- License Center
- Spatial Extender
- Tools Settings

The following Version 8 tools support Version 7 servers (with some restrictions) and Version 8 servers:
- Configuration Assistant (This tool has different components, of which only the import/export configuration file can be used with Version 7 servers; all of the components work with Version 8)
- Data Warehouse Center
- Replication Center
- Command Center (including the Web-version of this center)
- SQL Assist
- Development Center
- Visual Explain

In general, any Version 8 tool that is only launched from within the navigation tree of the Control Center, or any details view based on these tools, will not be available or accessible to Version 7 and earlier servers. You should consider using the Version 7 tools when working with Version 7 or earlier servers.

Scrollable cursor support

WIN	UNIX

Change: In Version 8, scrollable cursor functionality will not be supported from a Version 8 DB2 UDB for Unix and Windows client to a Version 7 DB2 UDB for Unix and Windows server. Support for scrollable cursors will only be available from a Version 8 DB2 UDB for Unix and Windows client to a DB2 UDB for Unix and Windows Version 8 server or to a DB2 UDB for z/OS and OS/390 Version 7 server. DB2 UDB for Unix and Windows Version 7 clients will continue to support existing scrollable cursor functionality to Version 8 DB2 UDB for Unix and Windows servers.

Resolution: Upgrade servers to Version 8.

Version 7 server access via a DB2 Connect Version 8 server

WIN	UNIX

Change: In Version 8, access from a DB2 UDB for Unix and Windows client to a Version 7 DB2 UDB server will not be supported through a Version 8 server, where the functionality is provided either by DB2 Connect Enterprise Edition Version 8 or by DB2 UDB Enterprise Server Edition Version 8.

Resolution: Upgrade servers to Version 8.

Messages

DB2 Connect messages returned instead of DB2 messages

WIN	UNIX

Change: In Version 8, conditions that would have returned a DB2 message in previous releases may now return a DB2 Connect message.

Examples:
- SQLCODE -30081 will be returned instead of SQLCODE -1224
- SQLCODE -30082 will be returned instead of SQLCODE -1403
- SQLCODE -30104 will be returned instead of SQLCODE -4930

Symptom: Applications coded to react to DB2 messages may not behave as before.

Configuration parameters

Obsolete database manager configuration parameters

WIN	UNIX

Change: The following database manager configuration parameters are obsolete:

- *backbufsz*: In previous versions you could perform a backup operation using a default buffer size, and the value of *backbufsz* would be taken as the default. In Version 8 you should explicitly specify the size of your backup buffers when you use the backup utility.
- *dft_client_adpt*: DCE directory services are no longer supported
- *dft_client_comm*: DCE directory services are no longer supported
- *dir_obj_name*: DCE directory services are no longer supported
- *dir_path_name*: DCE directory services are no longer supported
- *dir_type*: DCE directory services are no longer supported
- *dos_rqrioblk*
- *drda_heap_sz*
- *fcm_num_anchors*, *fcm_num_connect*, and *fcm_num_rqb*: DB2 will now adjust message anchors, connection entries, and request blocks dynamically and automatically, so you will not have to adjust these parameters
- *fileserver*: IPX/SPX is no longer supported
- *initdari_jvm*: Java stored procedures will now run multithreaded by default, and are run in separate processes from other language routines, so this parameter is no longer supported
- *ipx_socket*: IPX/SPX is no longer supported
- *jdk11_path*: replaced by *jdk_path* database manager configuration parameter
- *keepdari*: replaced by *keepfenced* database manager configuration parameter
- *max_logicagents*: replaced by *max_connections* database manager configuration parameter
- *maxdari*: replaced by *fenced_pool* database manager configuration parameter
- *num_initdaris*: replaced by *num_initfenced* database manager configuration parameter
- *objectname*: IPX/SPX is no longer supported

- *restbufsz*: In previous versions you could perform a restore operation using a default buffer size, and the value of *restbufsz* would be taken as the default. In Version 8 you should explicitly specify the size of your restore buffers when use restore utility.
- *route_obj_name*: DCE directory services are no longer supported
- *ss_logon*: this is an OS/2 parameter, and OS/2 is no longer supported
- *udf_mem_sz*: UDFs no longer pass data in shared memory, so this parameter is not supported

Resolution: Remove all references to these parameters from your applications.

Obsolete database configuration parameters

WIN	UNIX

Change: The following database configuration parameters are obsolete:
- *buffpage*: In previous versions, you could create or alter a buffer pool using a default size, and the value of *buffpage* would be taken as the default. In Version 8, you should explicitly specify the size of your buffer pools, using the SIZE keyword on the ALTER BUFFERPOOL or CREATE BUFFERPOOL statements.
- *copyprotect*
- *indexsort*

Resolution: Remove all references to these parameters from your applications.

Version 7 incompatibilities between releases

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. 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

db2set on AIX and Solaris

	UNIX	

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

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.

Appendix B. National language support (NLS)

This section contains information about the national language support (NLS) provided by DB2, including information about territories, 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 8 is available in Simplified Chinese, Traditional Chinese, Danish, English, Finnish, French, German, Italian, Japanese, Korean, Norwegian, Polish, Brazilian Portuguese, Russian, Spanish, and Swedish.

The DB2 Run-Time Client is available in these additional languages: Arabic, Bulgarian, Croatian, Czech, Dutch, Greek, Hebrew, Hungarian, Portuguese, Romanian, Slovakian, Slovenian, and Turkish.

Related reference:

- "Supported territory codes and code pages" on page 225

Supported territory codes and code pages

Table 23 on page 226 shows the languages and code sets supported by the database servers, and how these values are mapped to territory 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″), double-byte (″D″), or neutral (″N″). 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. However, if the group is neutral, then connection and conversion with any other code page listed is allowed.
- **Code set** shows the code set associated with the supported language. The code set is mapped to the DB2 code page.
- **Terr. ID** shows the territory identifier.
- **Territory 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.
- **Territory** shows the name of the region, country or countries.

Table 23. Supported languages and code sets

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
1200		16-bit Unicode			-	Any	Any
1208		UTF-8 encoding of Unicode				Any	Any
819	S-1	ISO8859-1	AL	355	sq_AL	AIX	Albania
850	S-1	IBM-850	AL	355	-	AIX	Albania
923	S-1	ISO8859-15	AL	355	sq_AL.8859-15	AIX	Albania
1208	N-1	UTF-8	AL	355	SQ_AL	AIX	Albania
37	S-1	IBM-37	AL	355	-	HOST	Albania
1140	S-1	IBM-1140	AL	355	-	HOST	Albania
819	S-1	iso88591	AL	355	-	HP	Albania
923	S-1	iso885915	AL	355	-	HP	Albania
1051	S-1	roman8	AL	355	-	HP	Albania
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	-	Sun	Albania
923	S-1	ISO8859-15	AL	355	-	Sun	Albania
1252	S-1	1252	AL	355	-	WIN	Albania
1046	S-6	IBM-1046	AA	785	Ar_AA	AIX	Arabic Countries/Regions
1089	S-6	ISO8859-6	AA	785	ar_AA	AIX	Arabic Countries/Regions
1208	N-1	UTF-8	AA	785	AR_AA	AIX	Arabic Countries/Regions
420	S-6	IBM-420	AA	785	-	HOST	Arabic Countries/Regions
425	S-6	IBM-425	AA	785	-	HOST	Arabic Countries/Regions
1089	S-6	iso88596	AA	785	ar_SA.iso88596	HP	Arabic Countries/Regions
864	S-6	IBM-864	AA	785	-	OS2	Arabic Countries/Regions
1256	S-6	1256	AA	785	-	WIN	Arabic Countries/Regions
819	S-1	ISO8859-1	AU	61	en_AU	AIX	Australia
850	S-1	IBM-850	AU	61	-	AIX	Australia
923	S-1	ISO8859-15	AU	61	en_AU.8859-15	AIX	Australia

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
1208	N-1	UTF-8	AU	61	EN_AU	AIX	Australia
37	S-1	IBM-37	AU	61	-	HOST	Australia
1140	S-1	IBM-1140	AU	61	-	HOST	Australia
819	S-1	iso88591	AU	61	-	HP	Australia
923	S-1	iso885915	AU	61	-	HP	Australia
1051	S-1	roman8	AU	61	-	HP	Australia
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	SCO	Australia
819	S-1	ISO8859-1	AU	61	en_AU	Sun	Australia
923	S-1	ISO8859-15	AU	61	-	Sun	Australia
1252	S-1	1252	AU	61	-	WIN	Australia
819	S-1	ISO8859-1	AT	43	-	AIX	Austria
850	S-1	IBM-850	AT	43	-	AIX	Austria
923	S-1	ISO8859-15	AT	43	-	AIX	Austria
1208	N-1	UTF-8	AT	43	-	AIX	Austria
37	S-1	IBM-37	AT	43	-	HOST	Austria
1140	S-1	IBM-1140	AT	43	-	HOST	Austria
819	S-1	iso88591	AT	43	-	HP	Austria
923	S-1	iso885915	AT	43	-	HP	Austria
1051	S-1	roman8	AT	43	-	HP	Austria
819	S-1	ISO-8859-1	AT	43	de_AT	Linux	Austria
923	S-1	ISO-8859-15	AT	43	de_AT@euro	Linux	Austria
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	de_AT	SCO	Austria
819	S-1	ISO8859-1	AT	43	de_AT	Sun	Austria
923	S-1	ISO8859-15	AT	43	de_AT.ISO8859-15	Sun	Austria
1252	S-1	1252	AT	43	-	WIN	Austria
915	S-5	ISO8859-5	BY	375	be_BY	AIX	Belarus
1208	N-1	UTF-8	BY	375	BE_BY	AIX	Belarus
1025	S-5	IBM-1025	BY	375	-	HOST	Belarus
1154	S-5	IBM-1154	BY	375	-	HOST	Belarus
915	S-5	ISO8859-5	BY	375	-	OS2	Belarus
1131	S-5	IBM-1131	BY	375	-	OS2	Belarus
1251	S-5	1251	BY	375	-	WIN	Belarus
819	S-1	ISO8859-1	BE	32	fr_BE	AIX	Belgium
819	S-1	ISO8859-1	BE	32	nl_BE	AIX	Belgium
850	S-1	IBM-850	BE	32	Fr_BE	AIX	Belgium

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
850	S-1	IBM-850	BE	32	Nl_BE	AIX	Belgium
923	S-1	ISO8859-15	BE	32	fr_BE.8859-15	AIX	Belgium
923	S-1	ISO8859-15	BE	32	nl_BE.8859-15	AIX	Belgium
1208	N-1	UTF-8	BE	32	FR_BE	AIX	Belgium
1208	N-1	UTF-8	BE	32	NL_BE	AIX	Belgium
274	S-1	IBM-274	BE	32	-	HOST	Belgium
500	S-1	IBM-500	BE	32	-	HOST	Belgium
1148	S-1	IBM-1148	BE	32	-	HOST	Belgium
819	S-1	iso88591	BE	32	-	HP	Belgium
923	S-1	iso885915	BE	32	-	HP	Belgium
819	S-1	ISO-8859-1	BE	32	fr_BE	Linux	Belgium
819	S-1	ISO-8859-1	BE	32	nl_BE	Linux	Belgium
923	S-1	ISO-8859-15	BE	32	fr_BE@euro	Linux	Belgium
923	S-1	ISO-8859-15	BE	32	nl_BE@euro	Linux	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	fr_BE	SCO	Belgium
819	S-1	ISO8859-1	BE	32	nl_BE	SCO	Belgium
819	S-1	ISO8859-1	BE	32	fr_BE	Sun	Belgium
819	S-1	ISO8859-1	BE	32	nl_BE	Sun	Belgium
923	S-1	ISO8859-15	BE	32	fr_BE.ISO8859-15	Sun	Belgium
923	S-1	ISO8859-15	BE	32	nl_BE.ISO8859-15	Sun	Belgium
1252	S-1	1252	BE	32	-	WIN	Belgium
915	S-5	ISO8859-5	BG	359	bg_BG	AIX	Bulgaria
1208	N-1	UTF-8	BG	359	BG_BG	AIX	Bulgaria
1025	S-5	IBM-1025	BG	359	-	HOST	Bulgaria
1154	S-5	IBM-1154	BG	359	-	HOST	Bulgaria
915	S-5	iso88595	BG	359	bg_BG.iso88595	HP	Bulgaria
855	S-5	IBM-855	BG	359	-	OS2	Bulgaria
915	S-5	ISO8859-5	BG	359	-	OS2	Bulgaria
1251	S-5	1251	BG	359	-	WIN	Bulgaria
819	S-1	ISO8859-1	BR	55	pt_BR	AIX	Brazil
850	S-1	IBM-850	BR	55	-	AIX	Brazil
923	S-1	ISO8859-15	BR	55	pt_BR.8859-15	AIX	Brazil
1208	N-1	UTF-8	BR	55	PT_BR	AIX	Brazil
37	S-1	IBM-37	BR	55	-	HOST	Brazil
1140	S-1	IBM-1140	BR	55	-	HOST	Brazil
819	S-1	ISO8859-1	BR	55	-	HP	Brazil
923	S-1	ISO8859-15	BR	55	-	HP	Brazil
819	S-1	ISO-8859-1	BR	55	pt_BR	Linux	Brazil

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
923	S-1	ISO-8859-15	BR	55	-	Linux	Brazil
850	S-1	IBM-850	BR	55	-	OS2	Brazil
819	S-1	ISO8859-1	BR	55	pt_BR	SCO	Brazil
819	S-1	ISO8859-1	BR	55	pt_BR	Sun	Brazil
923	S-1	ISO8859-15	BR	55	-	Sun	Brazil
1252	S-1	1252	BR	55	-	WIN	Brazil
819	S-1	ISO8859-1	CA	1	fr_CA	AIX	Canada
850	S-1	IBM-850	CA	1	Fr_CA	AIX	Canada
923	S-1	ISO8859-15	CA	1	fr_CA.8859-15	AIX	Canada
1208	N-1	UTF-8	CA	1	FR_CA	AIX	Canada
37	S-1	IBM-37	CA	1	-	HOST	Canada
1140	S-1	IBM-1140	CA	1	-	HOST	Canada
819	S-1	iso88591	CA	1	fr_CA.iso88591	HP	Canada
923	S-1	iso885915	CA	1	-	HP	Canada
1051	S-1	roman8	CA	1	fr_CA.roman8	HP	Canada
819	S-1	ISO-8859-1	CA	1	en_CA	Linux	Canada
923	S-1	ISO-8859-15	CA	1	-	Linux	Canada
850	S-1	IBM-850	CA	1	-	OS2	Canada
863	S-1	IBM-863	CA	2	-	OS2	Canada (French)
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
923	S-1	ISO8859-15	CA	1	-	Sun	Canada
1252	S-1	1252	CA	1	-	WIN	Canada
1383	D-4	IBM-eucCN	CN	86	zh_CN	AIX	China (PRC)
1386	D-4	GBK	CN	86	Zh_CN.GBK	AIX	China (PRC)
1208	N-1	UTF-8	CN	86	ZH_CN	AIX	China (PRC)
935	D-4	IBM-935	CN	86	-	HOST	China (PRC)
1388	D-4	IBM-1388	CN	86	-	HOST	China (PRC)
1383	D-4	hp15CN	CN	86	zh_CN.hp15CN	HP	China (PRC)
1383	D-4	GBK	CN	86	zh_CN.GBK	Linux	China (PRC)
1381	D-4	IBM-1381	CN	86	-	OS2	China (PRC)
1386	D-4	GBK	CN	86	-	OS2	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)
1208	N-1	UTF-8	CN	86	zh.UTF-8	Sun	China (PRC)
1381	D-4	IBM-1381	CN	86	-	WIN	China (PRC)
1386	D-4	GBK	CN	86	-	WIN	China (PRC)
1392/5488	D-4		CN	86	-		China (PRC)

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
See note 1 on page 243.							
912	S-2	ISO8859-2	HR	385	hr_HR	AIX	Croatia
1208	N-1	UTF-8	HR	385	HR_HR	AIX	Croatia
870	S-2	IBM-870	HR	385	-	HOST	Croatia
1153	S-2	IBM-1153	HR	385	-	HOST	Croatia
912	S-2	iso88592	HR	385	hr_HR.iso88592	HP	Croatia
912	S-2	ISO-8859-2	HR	385	hr_HR	Linux	Croatia
852	S-2	IBM-852	HR	385	-	OS2	Croatia
912	S-2	ISO8859-2	HR	385	hr_HR.ISO8859-2	SCO	Croatia
1250	S-2	1250	HR	385	-	WIN	Croatia
912	S-2	ISO8859-2	CZ	421	cs_CZ	AIX	Czech Republic
1208	N-1	UTF-8	CZ	421	CS_CZ	AIX	Czech Republic
870	S-2	IBM-870	CZ	421	-	HOST	Czech Republic
1153	S-2	IBM-1153	CZ	421	-	HOST	Czech Republic
912	S-2	iso88592	CZ	421	cs_CZ.iso88592	HP	Czech Republic
912	S-2	ISO-8859-2	CZ	421	cs_CZ	Linux	Czech Republic
852	S-2	IBM-852	CZ	421	-	OS2	Czech Republic
912	S-2	ISO8859-2	CZ	421	cs_CZ.ISO8859-2	SCO	Czech Republic
1250	S-2	1250	CZ	421	-	WIN	Czech Republic
819	S-1	ISO8859-1	DK	45	da_DK	AIX	Denmark
850	S-1	IBM-850	DK	45	Da_DK	AIX	Denmark
923	S-1	ISO8859-15	DK	45	da_DK.8859-15	AIX	Denmark
1208	N-1	UTF-8	DK	45	DA_DK	AIX	Denmark
277	S-1	IBM-277	DK	45	-	HOST	Denmark
1142	S-1	IBM-1142	DK	45	-	HOST	Denamrk
819	S-1	iso88591	DK	45	da_DK.iso88591	HP	Denmark
923	S-1	iso885915	DK	45	_	HP	Denmark
1051	S-1	roman8	DK	45	da_DK.roman8	HP	Denmark
819	S-1	ISO-8859-1	DK	45	da_DK	Linux	Denmark
923	S-1	ISO-8859-15	DK	45	-	Linux	Denmark
850	S-1	IBM-850	DK	45	-	OS2	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
923	S-1	ISO8859-15	DK	45	da.ISO8859-15	Sun	Denmark
1252	S-1	1252	DK	45	-	WIN	Denmark
922	S-10	IBM-922	EE	372	Et_EE	AIX	Estonia

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
1208	N-1	UTF-8	EE	372	ET_EE	AIX	Estonia
1122	S-10	IBM-1122	EE	372	-	HOST	Estonia
1157	S-10	IBM-1157	EE	372	-	HOST	Estonia
922	S-10	IBM-922	EE	372	-	OS2	Estonia
1257	S-10	1257	EE	372	-	WIN	Estonia
819	S-1	ISO8859-1	FI	358	fi_FI	AIX	Finland
850	S-1	IBM-850	FI	358	Fi_FI	AIX	Finland
923	S-1	ISO8859-15	FI	358	fi_FI.8859-15	AIX	Finland
1208	N-1	UTF-8	FI	358	FI_FI	AIX	Finland
278	S-1	IBM-278	FI	358	-	HOST	Finland
1143	S-1	IBM-1143	FI	358	-	HOST	Finland
819	S-1	iso88591	FI	358	fi_FI.iso88591	HP	Finland
923	S-1	iso885915	FI	358	-	HP	Finland
1051	S-1	roman8	FI	358	fi-FI.roman8	HP	Finland
819	S-1	ISO-8859-1	FI	358	fi_FI	Linux	Finland
923	S-1	ISO-8859-15	FI	358	fi_FI@euro	Linux	Finland
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	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
923	S-1	ISO8859-15	FI	358	fi.ISO8859-15	Sun	Finland
1252	S-1	1252	FI	358	-	WIN	Finland
915	S-5	ISO8859-5	MK	389	mk_MK	AIX	FYR Macedonia
1208	N-1	UTF-8	MK	389	MK_MK	AIX	FYR Macedonia
1025	S-5	IBM-1025	MK	389	-	HOST	FYR Macedonia
1154	S-5	IBM-1154	MK	389	-	HOST	FYR Macedonia
915	S-5	iso88595	MK	389	-	HP	FYR Macedonia
855	S-5	IBM-855	MK	389	-	OS2	FYR Macedonia
915	S-5	ISO8859-5	MK	389	-	OS2	FYR Macedonia
1251	S-5	1251	MK	389	-	WIN	FYR Macedonia
819	S-1	ISO8859-1	FR	33	fr_FR	AIX	France
850	S-1	IBM-850	FR	33	Fr_FR	AIX	France
923	S-1	ISO8859-15	FR	33	fr_FR.8859-15	AIX	France
1208	N-1	UTF-8	FR	33	FR_FR	AIX	France
297	S-1	IBM-297	FR	33	-	HOST	France
1147	S-1	IBM-1147	FR	33	-	HOST	France
819	S-1	iso88591	FR	33	fr_FR.iso88591	HP	France

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
923	S-1	iso885915	FR	33	-	HP	France
1051	S-1	roman8	FR	33	fr_FR.roman8	HP	France
819	S-1	ISO-8859-1	FR	33	fr_FR	Linux	France
923	S-1	ISO-8859-15	FR	33	fr_FR@euro	Linux	France
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	SCO	France
819	S-1	ISO8859-1	FR	33	fr_FR	SCO	France
819	S-1	ISO8859-1	FR	33	fr	Sun	France
923	S-1	ISO8859-15	FR	33	fr.ISO8859-15	Sun	France
1208	N-1	UTF-8	FR	33	fr.UTF-8	Sun	France
1252	S-1	1252	FR	33	-	WIN	France
819	S-1	ISO8859-1	DE	49	de_DE	AIX	Germany
850	S-1	IBM-850	DE	49	De_DE	AIX	Germany
923	S-1	ISO8859-15	DE	49	de_DE.8859-15	AIX	Germany
1208	N-1	UTF-8	DE	49	DE_DE	AIX	Germany
273	S-1	IBM-273	DE	49	-	HOST	Germany
1141	S-1	IBM-1141	DE	49	-	HOST	Germany
819	S-1	iso88591	DE	49	de_DE.iso88591	HP	Germany
923	S-1	iso885915	DE	49	_	HP	Germany
1051	S-1	roman8	DE	49	de_DE.roman8	HP	Germany
819	S-1	ISO-8859-1	DE	49	de_DE	Linux	Germany
923	S-1	ISO-8859-15	DE	49	de_DE@euro	Linux	Germany
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	SCO	Germany
819	S-1	ISO8859-1	DE	49	de_DE	SCO	Germany
819	S-1	ISO8859-1	DE	49	de	Sun	Germany
923	S-1	ISO8859-15	DE	49	de.ISO8859-15	Sun	Germany
1208	N-1	UTF-8	DE	49	de.UTF-8	Sun	Germany
1252	S-1	1252	DE	49	-	WIN	Germany
813	S-7	ISO8859-7	GR	30	el_GR	AIX	Greece
1208	N-1	UTF-8	GR	30	EL_GR	AIX	Greece
423	S-7	IBM-423	GR	30	-	HOST	Greece
875	S-7	IBM-875	GR	30	-	HOST	Greece
813	S-7	iso88597	GR	30	el_GR.iso88597	HP	Greece
813	S-7	ISO-8859-7	GR	30	el_GR	Linux	Greece
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.ISO8859-7	SCO	Greece

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
737	S-7	737	GR	30	-	WIN	Greece
1253	S-7	1253	GR	30	-	WIN	Greece
912	S-2	ISO8859-2	HU	36	hu_HU	AIX	Hungary
1208	N-1	UTF-8	HU	36	HU_HU	AIX	Hungary
870	S-2	IBM-870	HU	36	-	HOST	Hungary
1153	S-2	IBM-1153	HU	36	-	HOST	Hungary
912	S-2	iso88592	HU	36	hu_HU.iso88592	HP	Hungary
912	S-2	ISO-8859-2	HU	36	hu_HU	Linux	Hungary
852	S-2	IBM-852	HU	36	-	OS2	Hungary
912	S-2	ISO8859-2	HU	36	hu_HU.ISO8859-2	SCO	Hungary
1250	S-2	1250	HU	36	-	WIN	Hungary
819	S-1	ISO8859-1	IS	354	is_IS	AIX	Iceland
850	S-1	IBM-850	IS	354	Is_IS	AIX	Iceland
923	S-1	ISO8859-15	IS	354	is_IS.8859-15	AIX	Iceland
1208	N-1	UTF-8	IS	354	IS_IS	AIX	Iceland
871	S-1	IBM-871	IS	354	-	HOST	Iceland
1149	S-1	IBM-1149	IS	354	-	HOST	Iceland
819	S-1	iso88591	IS	354	is_IS.iso88591	HP	Iceland
923	S-1	iso885915	IS	354	-	HP	Iceland
1051	S-1	roman8	IS	354	is_IS.roman8	HP	Iceland
819	S-1	ISO-8859-1	IS	354	is_IS	Linux	Iceland
923	S-1	ISO-8859-15	IS	354	-	Linux	Iceland
850	S-1	IBM-850	IS	354	-	OS2	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
923	S-1	ISO8859-15	IS	354	-	Sun	Iceland
1252	S-1	1252	IS	354	-	WIN	Iceland
806	S-13	IBM-806	IN	91	hi_IN	-	India
1137	S-13	IBM-1137	IN	91	-	HOST	India
1252	S-1	1252	ID	62	-	WIN	Indonesia
819	S-1	ISO8859-1	IE	353	-	AIX	Ireland
850	S-1	IBM-850	IE	353	-	AIX	Ireland
923	S-1	ISO8859-15	IE	353	-	AIX	Ireland
1208	N-1	UTF-8	IE	353	-	AIX	Ireland
285	S-1	IBM-285	IE	353	-	HOST	Ireland
1146	S-1	IBM-1146	IE	353	-	HOST	Ireland

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
819	S-1	iso88591	IE	353	-	HP	Ireland
923	S-1	iso885915	IE	353	-	HP	Ireland
1051	S-1	roman8	IE	353	-	HP	Ireland
819	S-1	ISO-8859-1	IE	353	en_IE	Linux	Ireland
923	S-1	ISO-8859-15	IE	353	en_IE@euro	Linux	Ireland
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.ISO8859-1	SCO	Ireland
819	S-1	ISO8859-1	IE	353	en_IE	Sun	Ireland
923	S-1	ISO8859-15	IE	353	en_IE.ISO8859-15	Sun	Ireland
1252	S-1	1252	IE	353	-	WIN	Ireland
856	S-8	IBM-856	IL	972	Iw_IL	AIX	Israel
916	S-8	ISO8859-8	IL	972	iw_IL	AIX	Israel
1208	N-1	UTF-8	IL	972	HE-IL	AIX	Israel
916	S-8	ISO-8859-8	IL	972	iw_IL	Linux	Israel
424	S-8	IBM-424	IL	972	-	HOST	Israel
862	S-8	IBM-862	IL	972	-	OS2	Israel
1255	S-8	1255	IL	972	-	WIN	Israel
819	S-1	ISO8859-1	IT	39	it_IT	AIX	Italy
850	S-1	IBM-850	IT	39	It_IT	AIX	Italy
923	S-1	ISO8859-15	IT	39	it_IT.8859-15	AIX	Italy
1208	N-1	UTF-8	IT	39	It_IT	AIX	Italy
280	S-1	IBM-280	IT	39	-	HOST	Italy
1144	S-1	IBM-1144	IT	39	-	HOST	Italy
819	S-1	iso88591	IT	39	it_IT.iso88591	HP	Italy
923	S-1	iso885915	IT	39	_	HP	Italy
1051	S-1	roman8	IT	39	it_IT.roman8	HP	Italy
819	S-1	ISO-8859-1	IT	39	it_IT	Linux	Italy
923	S-1	ISO-8859-15	IT	39	it_IT@euro	Linux	Italy
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	SCO	Italy
819	S-1	ISO8859-1	IT	39	it_IT	SCO	Italy
819	S-1	ISO8859-1	IT	39	it	Sun	Italy
923	S-1	ISO8859-15	IT	39	it.ISO8859-15	Sun	Italy
1208	N-1	UTF-8	IT	39	it.UTF-8	Sun	Italy
1252	S-1	1252	IT	39	-	WIN	Italy
932	D-1	IBM-932	JP	81	Ja_JP	AIX	Japan
943	D-1	IBM-943	JP	81	Ja_JP	AIX	Japan

Table 23. Supported languages and code sets *(continued)*

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
See note 2 on page 244.							
954	D-1	IBM-eucJP	JP	81	ja_JP	AIX	Japan
1208	N-1	UTF-8	JP	81	JA_JP	AIX	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
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	EUC-JP	JP	81	ja_JP	Linux	Japan
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	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
943	D-1	IBM-943	JP	81	ja_JP.PCK	Sun	Japan
954	D-1	eucJP	JP	81	ja	Sun	Japan
954	D-1	eucJP	JP	81	japanese	Sun	Japan
1208	N-1	UTF-8	JP	81	ja_JP.UTF-8	Sun	Japan
943	D-1	IBM-943	JP	81	-	WIN	Japan
1394	D-1		JP	81	-		Japan
See note 3 on page 244.							
1251	S-5	1251	KZ	7	-	WIN	Kazakhstan
970	D-3	IBM-eucKR	KR	82	ko_KR	AIX	Korea, South
1208	N-1	UTF-8	KR	82	KO_KR	AIX	Korea, South
933	D-3	IBM-933	KR	82	-	HOST	Korea, South
1364	D-3	IBM-1364	KR	82	-	HOST	Korea, South
970	D-3	eucKR	KR	82	ko_KR.eucKR	HP	Korea, South
970	D-3	EUC-KR	KR	82	ko_KR	Linux	Korea, South
949	D-3	IBM-949	KR	82	-	OS2	Korea, South
970	D-3	eucKR	KR	82	ko_KR.eucKR	SGI	Korea, South
970	D-3	5601	KR	82	ko	Sun	Korea, South
1208	N-1	UTF-8	KR	82	ko.UTF-8	Sun	Korea, South
1363	D-3	1363	KR	82	-	WIN	Korea, South
819	S-1	ISO8859-1	Lat	3	-	AIX	Latin America

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
850	S-1	IBM-850	Lat	3	-	AIX	Latin America
923	S-1	ISO8859-15	Lat	3	-	AIX	Latin America
1208	N-1	UTF-8	Lat	3	-	AIX	Latin America
284	S-1	IBM-284	Lat	3	-	HOST	Latin America
1145	S-1	IBM-1145	Lat	3	-	HOST	Latin America
819	S-1	iso88591	Lat	3	-	HP	Latin America
923	S-1	iso885915	Lat	3	-	HP	Latin America
1051	S-1	roman8	Lat	3	-	HP	Latin America
819	S-1	ISO-8859-1	Lat	3	-	Linux	Latin America
923	S-1	ISO-8859-15	Lat	3	-	Linux	Latin America
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	-	Sun	Latin America
923	S-1	ISO8859-15	Lat	3	-	Sun	Latin America
1252	S-1	1252	Lat	3	-	WIN	Latin America
921	S-10	IBM-921	LV	371	Lv_LV	AIX	Latvia
1208	N-1	UTF-8	LV	371	LV_LV	AIX	Latvia
1112	S-10	IBM-1112	LV	371	-	HOST	Latvia
1156	S-10	IBM-1156	LV	371	-	HOST	Latvia
921	S-10	IBM-921	LV	371	-	OS2	Latvia
1257	S-10	1257	LV	371	-	WIN	Latvia
921	S-10	IBM-921	LT	370	Lt_LT	AIX	Lithuania
1208	N-1	UTF-8	LT	370	LT_LT	AIX	Lithuania
1112	S-10	IBM-1112	LT	370	-	HOST	Lithuania
1156	S-10	IBM-1156	LT	370	-	HOST	Lithuania
921	S-10	IBM-921	LT	370	-	OS2	Lithuania
1257	S-10	1257	LT	370	-	WIN	Lithuania
1252	S-1	1252	ID	60	-	WIN	Malaysia
819	S-1	ISO8859-1	NL	31	nl_NL	AIX	Netherlands
850	S-1	IBM-850	NL	31	Nl_NL	AIX	Netherlands
923	S-1	ISO8859-15	NL	31	nl_NL.8859-15	AIX	Netherlands
1208	N-1	UTF-8	NL	31	NL_NL	AIX	Netherlands
37	S-1	IBM-37	NL	31	-	HOST	Netherlands
1140	S-1	IBM-1140	NL	31	-	HOST	Netherlands
819	S-1	iso88591	NL	31	nl_NL.iso88591	HP	Netherlands
923	S-1	iso885915	NL	31	_	HP	Netherlands
1051	S-1	roman8	NL	31	nl_NL.roman8	HP	Netherlands
819	S-1	ISO-8859-1	NL	31	nl_NL	Linux	Netherlands

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
923	S-1	ISO-8859-15	NL	31	nl_NL@euro	Linux	Netherlands
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	SCO	Netherlands
819	S-1	ISO8859-1	NL	31	nl_NL	SCO	Netherlands
819	S-1	ISO8859-1	NL	31	nl	Sun	Netherlands
923	S-1	ISO8859-15	NL	31	nl.ISO8859-15	Sun	Netherlands
1252	S-1	1252	NL	31	-	WIN	Netherlands
819	S-1	ISO8859-1	NZ	64	-	AIX	New Zealand
850	S-1	IBM-850	NZ	64	-	AIX	New Zealand
923	S-1	ISO8859-15	NZ	64	-	AIX	New Zealand
1208	N-1	UTF-8	NZ	64	-	AIX	New Zealand
37	S-1	IBM-37	NZ	64	-	HOST	New Zealand
1140	S-1	IBM-1140	NZ	64	-	HOST	New Zealand
819	S-1	ISO8859-1	NZ	64	-	HP	New Zealand
923	S-1	ISO8859-15	NZ	64	-	HP	New Zealand
850	S-1	IBM-850	NZ	64	-	OS2	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
923	S-1	ISO8859-15	NZ	64	-	Sun	New Zealand
1252	S-1	1252	NZ	64	-	WIN	New Zealand
819	S-1	ISO8859-1	NO	47	no_NO	AIX	Norway
850	S-1	IBM-850	NO	47	No_NO	AIX	Norway
923	S-1	ISO8859-15	NO	47	no_NO.8859-15	AIX	Norway
1208	N-1	UTF-8	NO	47	NO_NO	AIX	Norway
277	S-1	IBM-277	NO	47	-	HOST	Norway
1142	S-1	IBM-1142	NO	47	-	HOST	Norway
819	S-1	iso88591	NO	47	no_NO.iso88591	HP	Norway
923	S-1	iso885915	NO	47	-	HP	Norway
1051	S-1	roman8	NO	47	no_NO.roman8	HP	Norway
819	S-1	ISO-8859-1	NO	47	no_NO	Linux	Norway
923	S-1	ISO-8859-15	NO	47	-	Linux	Norway
850	S-1	IBM-850	NO	47	-	OS2	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
923	S-1	ISO8859-15	NO	47	-	Sun	Norway
1252	S-1	1252	NO	47	-	WIN	Norway
912	S-2	ISO8859-2	PL	48	pl_PL	AIX	Poland

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
1208	N-1	UTF-8	PL	48	PL_PL	AIX	Poland
870	S-2	IBM-870	PL	48	-	HOST	Poland
1153	S-2	IBM-1153	PL	48	-	HOST	Poland
912	S-2	iso88592	PL	48	pl_PL.iso88592	HP	Poland
912	S-2	ISO-8859-2	PL	48	pl_PL	Linux	Poland
852	S-2	IBM-852	PL	48	-	OS2	Poland
912	S-2	ISO8859-2	PL	48	pl_PL.ISO8859-2	SCO	Poland
1250	S-2	1250	PL	48	-	WIN	Poland
819	S-1	ISO8859-1	PT	351	pt_PT	AIX	Portugal
850	S-1	IBM-850	PT	351	Pt_PT	AIX	Portugal
923	S-1	ISO8859-15	PT	351	pt_PT.8859-15	AIX	Portugal
1208	N-1	UTF-8	PT	351	PT_PT	AIX	Portugal
37	S-1	IBM-37	PT	351	-	HOST	Portugal
1140	S-1	IBM-1140	PT	351	-	HOST	Portugal
819	S-1	iso88591	PT	351	pt_PT.iso88591	HP	Portugal
923	S-1	iso885915	PT	351	-	HP	Portugal
1051	S-1	roman8	PT	351	pt_PT.roman8	HP	Portugal
819	S-1	ISO-8859-1	PT	351	pt_PT	Linux	Portugal
923	S-1	ISO-8859-15	PT	351	pt_PT@euro	Linux	Portugal
850	S-1	IBM-850	PT	351	-	OS2	Portugal
860	S-1	IBM-860	PT	351	-	OS2	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
923	S-1	ISO8859-15	PT	351	pt.ISO8859-15	Sun	Portugal
1252	S-1	1252	PT	351	-	WIN	Portugal
912	S-2	ISO8859-2	RO	40	ro_RO	AIX	Romania
1208	N-1	UTF-8	RO	40	RO_RO	AIX	Romania
870	S-2	IBM-870	RO	40	-	HOST	Romania
1153	S-2	IBM-1153	RO	40	-	HOST	Romania
912	S-2	iso88592	RO	40	ro_RO.iso88592	HP	Romania
912	S-2	ISO-8859-2	RO	40	ro_RO	Linux	Romania
852	S-2	IBM-852	RO	40	-	OS2	Romania
912	S-2	ISO8859-2	RO	40	ro_RO.ISO8859-2	SCO	Romania
1250	S-2	1250	RO	40	-	WIN	Romania
915	S-5	ISO8859-5	RU	7	ru_RU	AIX	Russia
1208	N-1	UTF-8	RU	7	RU_RU	AIX	Russia
1025	S-5	IBM-1025	RU	7	-	HOST	Russia
1154	S-5	IBM-1154	RU	7	-	HOST	Russia

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
915	S-5	iso88595	RU	7	ru_RU.iso88595	HP	Russia
915	S-5	ISO-8859-5	RU	7	ru_RU	Linux	Russia
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.ISO8859-5	SCO	Russia
1251	S-5	1251	RU	7	-	WIN	Russia
915	S-5	ISO8859-5	SP	381	sr_SP	AIX	Serbia/Montenegro
1208	N-1	UTF-8	SP	381	SR_SP	AIX	Serbia/Montenegro
1025	S-5	IBM-1025	SP	381	-	HOST	Serbia/Montenegro
1154	S-5	IBM-1154	SP	381	-	HOST	Serbia/Montenegro
915	S-5	iso88595	SP	381	-	HP	Serbia/Montenegro
855	S-5	IBM-855	SP	381	-	OS2	Serbia/Montenegro
915	S-5	ISO8859-5	SP	381	-	OS2	Serbia/Montenegro
1251	S-5	1251	SP	381	-	WIN	Serbia/Montenegro
912	S-2	ISO8859-2	SK	422	sk_SK	AIX	Slovakia
1208	N-1	UTF-8	SK	422	SK_SK	AIX	Slovakia
870	S-2	IBM-870	SK	422	-	HOST	Slovakia
1153	S-2	IBM-1153	SK	422	-	HOST	Slovakia
912	S-2	iso88592	SK	422	sk_SK.iso88592	HP	Slovakia
852	S-2	IBM-852	SK	422	-	OS2	Slovakia
912	S-2	ISO8859-2	SK	422	sk_SK.ISO8859-2	SCO	Slovakia
1250	S-2	1250	SK	422	-	WIN	Slovakia
912	S-2	ISO8859-2	SI	386	sl_SI	AIX	Slovenia
1208	N-1	UTF-8	SI	386	SL_SI	AIX	Slovenia
870	S-2	IBM-870	SI	386	-	HOST	Slovenia
1153	S-2	IBM-1153	SI	386	-	HOST	Slovenia
912	S-2	iso88592	SI	386	sl_SI.iso88592	HP	Slovenia
912	S-2	ISO-8859-2	SI	386	sl_SI	Linux	Slovenia
852	S-2	IBM-852	SI	386	-	OS2	Slovenia
912	S-2	ISO8859-2	SI	386	sl_SI.ISO8859-2	SCO	Slovenia
1250	S-2	1250	SI	386	-	WIN	Slovenia
819	S-1	ISO8859-1	ZA	27	en_ZA	AIX	South Africa
850	S-1	IBM-850	ZA	27	En_ZA	AIX	South Africa
923	S-1	ISO8859-15	ZA	27	en_ZA.8859-15	AIX	South Africa
1208	N-1	UTF-8	ZA	27	EN_ZA	AIX	South Africa
285	S-1	IBM-285	ZA	27	-	HOST	South Africa
1146	S-1	IBM-1146	ZA	27	-	HOST	South Africa
819	S-1	iso88591	ZA	27	-	HP	South Africa

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
923	S-1	iso885915	ZA	27	-	HP	South Africa
1051	S-1	roman8	ZA	27	-	HP	South Africa
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.ISO8859-1	SCO	South Africa
819	S-1	ISO8859-1	ZA	27	-	Sun	South Africa
923	S-1	ISO8859-15	ZA	27	-	Sun	South Africa
1252	S-1	1252	ZA	27	-	WIN	South Africa
819	S-1	ISO8859-1	ES	34	es_ES	AIX	Spain
819	S-1	ISO8859-1	ES	34	ca_ES	AIX	Spain (Catalan)
850	S-1	IBM-850	ES	34	Es_ES	AIX	Spain
850	S-1	IBM-850	ES	34	Ca_ES	AIX	Spain (Calalan)
923	S-1	ISO8859-15	ES	34	es_ES.8859-15	AIX	Spain
923	S-1	ISO8859-15	ES	34	ca_ES.8859-15	AIX	Spain (Catalan)
1208	N-1	UTF-8	ES	34	ES_ES	AIX	Spain
1208	N-1	UTF-8	ES	34	CA_ES	AIX	Spain (Catalan)
284	S-1	IBM-284	ES	34	-	HOST	Spain
1145	S-1	IBM-1145	ES	34	-	HOST	Spain
819	S-1	iso88591	ES	34	es_ES.iso88591	HP	Spain
923	S-1	iso885915	ES	34	-	HP	Spain
1051	S-1	roman8	ES	34	es_ES.roman8	HP	Spain
819	S-1	ISO-8859-1	ES	34	es_ES	Linux	Spain
923	S-1	ISO-8859-15	ES	34	es_ES@euro	Linux	Spain
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	SCO	Spain
819	S-1	ISO8859-1	ES	34	es_ES	SCO	Spain
819	S-1	ISO8859-1	ES	34	es	Sun	Spain
923	S-1	ISO8859-15	ES	34	es.ISO8859-15	Sun	Spain
1208	N-1	UTF-8	ES	34	es.UTF-8	Sun	Spain
1252	S-1	1252	ES	34	-	WIN	Spain
819	S-1	ISO8859-1	SE	46	sv_SE	AIX	Sweden
850	S-1	IBM-850	SE	46	Sv_SE	AIX	Sweden
923	S-1	ISO8859-15	SE	46	sv_SE.8859-15	AIX	Sweden
1208	N-1	UTF-8	SE	46	SV_SE	AIX	Sweden
278	S-1	IBM-278	SE	46	-	HOST	Sweden
1143	S-1	IBM-1143	SE	46	-	HOST	Sweden
819	S-1	iso88591	SE	46	sv_SE.iso88591	HP	Sweden
923	S-1	iso885915	SE	46	-	HP	Sweden
1051	S-1	roman8	SE	46	sv_SE.roman8	HP	Sweden

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
819	S-1	ISO-8859-1	SE	46	sv_SE	Linux	Sweden
923	S-1	ISO-8859-15	SE	46	-	Linux	Sweden
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	SCO	Sweden
819	S-1	ISO8859-1	SE	46	sv_SE	SCO	Sweden
819	S-1	ISO8859-1	SE	46	sv	Sun	Sweden
923	S-1	ISO8859-15	SE	46	sv.ISO8859-15	Sun	Sweden
1208	N-1	UTF-8	SE	46	sv.UTF-8	Sun	Sweden
1252	S-1	1252	SE	46	-	WIN	Sweden
819	S-1	ISO8859-1	CH	41	de_CH	AIX	Switzerland
850	S-1	IBM-850	CH	41	De_CH	AIX	Switzerland
923	S-1	ISO8859-15	CH	41	de_CH.8859-15	AIX	Switzerland
1208	N-1	UTF-8	CH	41	DE_CH	AIX	Switzerland
500	S-1	IBM-500	CH	41	-	HOST	Switzerland
1148	S-1	IBM-1148	CH	41	-	HOST	Switzerland
819	S-1	iso88591	CH	41	-	HP	Switzerland
923	S-1	iso885915	CH	41	-	HP	Switzerland
1051	S-1	roman8	CH	41	-	HP	Switzerland
819	S-1	ISO-8859-1	CH	41	de_CH	Linux	Switzerland
923	S-1	ISO-8859-15	CH	41	-	Linux	Switzerland
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	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
923	S-1	ISO8859-15	CH	41	-	Sun	Switzerland
1252	S-1	1252	CH	41	-	WIN	Switzerland
950	D-2	big5	TW	88	Zh_TW	AIX	Taiwan
964	D-2	IBM-eucTW	TW	88	zh_TW	AIX	Taiwan
1208	N-1	UTF-8	TW	88	ZH_TW	AIX	Taiwan
937	D-2	IBM-937	TW	88	-	HOST	Taiwan
1371	D-2	IBM-1371	TW	88	-	HOST	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	zh_TW	Linux	Taiwan
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

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
950	D-2	big5	TW	88	zh_TW.BIG5	Sun	Taiwan
See note 8 on page 245.							
964	D-2	cns11643	TW	88	zh_TW	Sun	Taiwan
1208	N-1	UTF-8	TW	88	zh_TW.UTF-8	Sun	Taiwan
950	D-2	big5	TW	88	-	WIN	Taiwan
874	S-20	TIS620-1	TH	66	th_TH	AIX	Thailand
1208	N-1	UTF-8	TH	66	TH_TH	AIX	Thailand
838	S-20	IBM-838	TH	66	-	HOST	Thailand
1160	S-20	IBM-1160	TH	66	-	HOST	Thailand
874	S-20	tis620	TH	66	th_TH.tis620	HP	Thailand
874	S-20	TIS620-1	TH	66	-	OS2	Thailand
874	S-20	TIS620-1	TH	66	-	WIN	Thailand
920	S-9	ISO8859-9	TR	90	tr_TR	AIX	Turkey
1208	N-1	UTF-8	TR	90	TR_TR	AIX	Turkey
1026	S-9	IBM-1026	TR	90	-	HOST	Turkey
1155	S-9	IBM-1155	TR	90	-	HOST	Turkey
920	S-9	iso88599	TR	90	tr_TR.iso88599	HP	Turkey
920	S-9	ISO-8859-9	TR	90	tr_TR	Linux	Turkey
857	S-9	IBM-857	TR	90	-	OS2	Turkey
920	S-9	ISO8859-9	TR	90	tr_TR.ISO8859-9	SCO	Turkey
1254	S-9	1254	TR	90	-	WIN	Turkey
819	S-1	ISO8859-1	GB	44	en_GB	AIX	U.K.
850	S-1	IBM-850	GB	44	En_GB	AIX	U.K.
923	S-1	ISO8859-15	GB	44	en_GB.8859-15	AIX	U.K.
1208	N-1	UTF-8	GB	44	EN_GB	AIX	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	iso88591	GB	44	en_GB.iso88591	HP	U.K.
923	S-1	iso885915	GB	44	-	HP	U.K.
1051	S-1	roman8	GB	44	en_GB.roman8	HP	U.K.
819	S-1	ISO-8859-1	GB	44	en_GB	Linux	U.K.
923	S-1	ISO-8859-15	GB	44	-	Linux	U.K.
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	SCO	U.K.
819	S-1	ISO8859-1	GB	44	en	SCO	U.K.
819	S-1	ISO8859-1	GB	44	en_GB	Sun	U.K.
923	S-1	ISO8859-15	GB	44	en_GB.ISO8859-15	Sun	U.K.
1252	S-1	1252	GB	44	-	WIN	U.K.

Table 23. Supported languages and code sets (continued)

Code page	Group	Code set	Terr. ID	Territory code	Locale	OS	Territory
1124	S-12	IBM-1124	UA	380	Uk_UA	AIX	Ukraine
1208	N-1	UTF-8	UA	380	UK_UA	AIX	Ukraine
1123	S-12	IBM-1123	UA	380	-	HOST	Ukraine
1158	S-12	IBM-1158	UA	380	-	HOST	Ukraine
1125	S-12	IBM-1125	UA	380	-	OS2	Ukraine
1251	S-12	1251	UA	380	-	WIN	Ukraine
819	S-1	ISO8859-1	US	1	en_US	AIX	USA
850	S-1	IBM-850	US	1	En_US	AIX	USA
923	S-1	ISO8859-15	US	1	en_US.8859-15	AIX	USA
1208	N-1	UTF-8	US	1	EN_US	AIX	USA
37	S-1	IBM-37	US	1	-	HOST	USA
1140	S-1	IBM-1140	US	1	-	HOST	USA
819	S-1	iso88591	US	1	en_US.iso88591	HP	USA
923	S-1	iso885915	US	1	-	HP	USA
1051	S-1	roman8	US	1	en_US.roman8	HP	USA
819	S-1	ISO-8859-1	US	1	en_US	Linux	USA
923	S-1	ISO-8859-15	US	1	-	Linux	USA
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	SCO	USA
819	S-1	ISO8859-1	US	1	en_US	SGI	USA
819	S-1	ISO8859-1	US	1	en_US	Sun	USA
923	S-1	ISO8859-15	US	1	en_US.ISO8859-15	Sun	USA
1208	N-1	UTF-8	US	1	en_US.UTF-8	Sun	USA
1252	S-1	1252	US	1	-	WIN	USA
1129	S-11	IBM-1129	VN	84	Vi_VN	AIX	Vietnam
1208	N-1	UTF-8	VN	84	VI_VN	AIX	Vietnam
1130	S-11	IBM-1130	VN	84	-	HOST	Vietnam
1164	S-11	IBM-1164	VN	84	-	HOST	Vietnam
1129	S-11	IBM-1129	VN	84	-	OS2	Vietnam
1258	S-11	1258	VN	84	-	WIN	Vietnam

Notes:

1. CCSIDs 1392 and 5488 (GB 18030) can only be used with the load or import utilities to move data from CCSIDs 1392 and 5488 to a DB2 Unicode database, or to export from a DB2 Unicode database to CCSIDs 1392 or 5488.

2. 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.
3. Code page 1394 (Shift JIS X0213) 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.
4. The following map to Arabic Countries/Regions (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)
5. The following map to English (US):
   - English (Jamaica)
   - English (Caribbean)
6. 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)

7. The following Indic scripts are supported through Unicode: Hindi, Gujarati, Kannada, Konkani, Marathi, Punjabi, Sanskrit, Tamil and Telugu.
8. Both Microsoft code page 950 and Solaris code page 950, also known as Big5, do not support the following characters in IBM 950:

Code range	Description	Solaris Big5	Microsoft Big5	IBM Big5
X'C6A1'-X'C8FE'	Symbols	Reserved area	User-defined characters	Symbols
X'F9D6'-X'F9FE'	ETen extension	Reserved area	System fonts	UDC
X'F286'-X'F9A0'	IBM selected characters	Reserved area	Not used	IBM selected characters

Enabling and disabling euro symbol support

DB2 Universal Database provides support for the euro currency symbol. The euro symbol has been added to numerous code pages. Many DB2 UDB internal code page conversion tables, and many external code page conversion table files located in the `sqllib/conv/` directory have been enhanced to support the euro symbol.

Microsoft ANSI code pages have been modified to include the euro currency symbol in position X'80'. Code page 850 has been modified to replace the character DOTLESS I (found at position X'D5') with the euro currency symbol. DB2 UDB internal code page conversion routines use these revised code page definitions as the default to provide euro symbol support.

However, if you want to use the non-euro definitions of the code page conversion tables, follow the procedure below after installation is complete.

Prerequisites:

For replacing existing external code page conversion table files, you may want to back up the current files before copying the non-euro versions over them.

The files are located in the directory `sqllib/conv/`. On UNIX, `sqllib/conv/` is linked to the install path of DB2.

Procedure:

To disable euro-symbol support:

1. Download the appropriate conversion table files, in binary, from ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/conv/. This ftp server is anonymous, so if you are connecting via the command line, log in as user ″anonymous″ and use your e-mail address as your password. After logging in, change to the conversion tables directory: `cd ps/products/db2/info/vr8/conv`
2. Copy the files to your `sqllib/conv/` directory.
3. Restart DB2.

Code pages 819 and 1047:

For code pages 819 (ISO 8859-1 Latin 1 ASCII) and 1047 (Latin 1 Open System EBCDIC), the euro replacement code pages, 923 (ISO 8859-15 Latin 9 ASCII) and 924 (Latin 9 Open System EBCDIC) respectively, contain not just the euro symbol but also several new characters. DB2 continues to use the old (non-euro) definitions of these two code pages and conversion tables, namely 819 and 1047, by default. There are two ways to activate the new 923/924 code page and the associated conversion tables:

- Create a new database that uses the new code page. For example,
  ```
  DB2 CREATE DATABASE dbname USING CODESET ISO8859-15 TERRITORY US
  ```
- Rename or copy the 923 or 924 conversion table file in the `sqllib/conv/` directory to 819 or 1047, respectively.

Related concepts:

- “Character conversion” in the *SQL Reference, Volume 1*

Related reference:

- “Conversion tables for code pages 923 and 924” on page 256
- “Conversion table files for euro-enabled code pages” on page 246

Conversion table files for euro-enabled code pages

The following tables list the conversion tables that have been enhanced to support the euro currency symbol. If you want to disable euro symbol support, download the conversion table file indicated in the column titled ″Conversion table file″.

Arabic:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
420, 16804	1046, 9238	04201046.cnv, IBM00420.ucs

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
420, 16804	1256, 5352	04201256.cnv, IBM00420.ucs
420, 16804	1200, 1208, 13488, 17584	IBM00420.ucs
864, 17248	1046, 9238	08641046.cnv, 10460864.cnv, IBM00864.ucs
864, 17248	1256, 5352	08641256.cnv, 12560864.cnv, IBM00864.ucs
864, 17248	1200, 1208, 13488, 17584	IBM00864.ucs
1046, 9238	864, 17248	10460864.cnv, 08641046.cnv, IBM01046.ucs
1046, 9238	1089	10461089.cnv, 10891046.cnv, IBM01046.ucs
1046, 9238	1256, 5352	10461256.cnv, 12561046.cnv, IBM01046.ucs
1046, 9238	1200, 1208, 13488, 17584	IBM01046.ucs
1089	1046, 9238	10891046.cnv, 10461089.cnv
1256, 5352	864, 17248	12560864.cnv, 08641256.cnv, IBM01256.ucs
1256, 5352	1046, 9238	12561046.cnv, 10461256.cnv, IBM01256.ucs
1256, 5352	1200, 1208, 13488, 17584	IBM01256.ucs

Baltic:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
921, 901	1257	09211257.cnv, 12570921.cnv, IBM00921.ucs
921, 901	1200, 1208, 13488, 17584	IBM00921.ucs
1112, 1156	1257, 5353	11121257.cnv
1257, 5353	921, 901	12570921.cnv, 09211257.cnv, IBM01257.ucs
1257, 5353	922, 902	12570922.cnv, 09221257.cnv, IBM01257.ucs
1257, 5353	1200, 1208, 13488, 17584	IBM01257.ucs

Belarus:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
1131, 849	1251, 5347	11311251.cnv, 12511131.cnv
1131, 849	1283	11311283.cnv

Cyrillic:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
855, 872	866, 808	08550866.cnv, 08660855.cnv
855, 872	1251, 5347	08551251.cnv, 12510855.cnv
866, 808	855, 872	08660855.cnv, 08550866.cnv
866, 808	1251, 5347	08661251.cnv, 12510866.cnv
1025, 1154	855, 872	10250855.cnv, IBM01025.ucs
1025, 1154	866, 808	10250866.cnv, IBM01025.ucs
1025, 1154	1131, 849	10251131.cnv, IBM01025.ucs
1025, 1154	1251, 5347	10251251.cnv, IBM01025.ucs
1025, 1154	1200, 1208, 13488, 17584	IBM01025.ucs
1251, 5347	855, 872	12510855.cnv, 08551251.cnv, IBM01251.ucs
1251, 5347	866, 808	12510866.cnv, 08661251.cnv, IBM01251.ucs
1251, 5347	1124	12511124.cnv, 11241251.cnv, IBM01251.ucs
1251, 5347	1125, 848	12511125.cnv, 11251251.cnv, IBM01251.ucs
1251, 5347	1131, 849	12511131.cnv, 11311251.cnv, IBM01251.ucs
1251, 5347	1200, 1208, 13488, 17584	IBM01251.ucs

Estonia:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
922, 902	1257	09221257.cnv, 12570922.cnv, IBM00922.ucs
922, 902	1200, 1208, 13488, 17584	IBM00922.ucs

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
1122, 1157	1257, 5353	11221257.cnv

Greek:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
423	869, 9061	04230869.cnv
813, 4909	869, 9061	08130869.cnv, 08690813.cnv, IBM00813.ucs
813, 4909	1253, 5349	08131253.cnv, 12530813.cnv, IBM00813.ucs
813, 4909	1200, 1208, 13488, 17584	IBM00813.ucs
869, 9061	813, 4909	08690813.cnv, 08130869.cnv
869, 9061	1253, 5349	08691253.cnv, 12530869.cnv
875, 4971	813, 4909	08750813.cnv, IBM00875.ucs
875, 4971	1253, 5349	08751253.cnv, IBM00875.ucs
875, 4971	1200, 1208, 13488, 17584	IBM00875.ucs
1253, 5349	813, 4909	12530813.cnv, 08131253.cnv, IBM01253.ucs
1253, 5349	869, 9061	12530869.cnv, 08691253.cnv, IBM01253.ucs
1253, 5349	1200, 1208, 13488, 17584	IBM01253.ucs

Hebrew:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
424, 12712	856, 9048	04240856.cnv, IBM00424.ucs
424, 12712	862, 867	04240862.cnv, IBM00424.ucs
424, 12712	916	04240916.cnv, IBM00424.ucs
424, 12712	1255, 5351	04241255.cnv, IBM00424.ucs
424, 12712	1200, 1208, 13488, 17584	IBM00424.ucs
856, 9048	862, 867	08560862.cnv, 08620856.cnv, IBM0856.ucs
856, 9048	916	08560916.cnv, 09160856.cnv, IBM0856.ucs

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
856, 9048	1255, 5351	08561255.cnv, 12550856.cnv, IBM0856.ucs
856, 9048	1200, 1208, 13488, 17584	IBM0856.ucs
862, 867	856, 9048	08620856.cnv, 08560862.cnv, IBM00862.ucs
862, 867	916	08620916.cnv, 09160862.cnv, IBM00862.ucs
862, 867	1255, 5351	08621255.cnv, 12550862.cnv, IBM00862.ucs
862, 867	1200, 1208, 13488, 17584	IBM00862.ucs
916	856, 9048	09160856.cnv, 08560916.cnv
916	862, 867	09160862.cnv, 08620916.cnv
1255, 5351	856, 9048	12550856.cnv, 08561255.cnv, IBM01255.ucs
1255, 5351	862, 867	12550862.cnv, 08621255.cnv, IBM01255.ucs
1255, 5351	1200, 1208, 13488, 17584	IBM01255.ucs

Japanese:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
290, 8482	850, 858	02900850.cnv
954	1200, 1208, 13488, 17584	0954ucs2.cnv, ucs20954.cnv
1027, 5123	850, 858	10270850.cnv

Latin-1:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
37, 1140	437	00370437.cnv, IBM00037.ucs
37, 1140	850, 858	00370850.cnv, IBM00037.ucs
37, 1140	860	00370860.cnv, IBM00037.ucs
37, 1140	1051	00371051.cnv, IBM00037.ucs
37, 1140	1252, 5348	00371252.cnv, IBM00037.ucs
37, 1140	1275	00371275.cnv, IBM00037.ucs

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
37, 1140	1200, 1208, 13488, 17584	IBM00037.ucs
273, 1141	437	02730437.cnv, IBM00273.ucs
273, 1141	850, 858	02730850.cnv, IBM00273.ucs
273, 1141	1051	02731051.cnv, IBM00273.ucs
273, 1141	1252, 5348	02731252.cnv, IBM00273.ucs
273, 1141	1275	02731275.cnv, IBM00273.ucs
273, 1141	1200, 1208, 13488, 17584	IBM00273.ucs
277, 1142	437	02770437.cnv, IBM00277.ucs
277, 1142	850, 858	02770850.cnv, IBM00277.ucs
277, 1142	1051	02771051.cnv, IBM00277.ucs
277, 1142	1252, 5348	02771252.cnv, IBM00277.ucs
277, 1142	1275	02771275.cnv, IBM00277.ucs
277, 1142	1200, 1208, 13488, 17584	IBM00277.ucs
278, 1143	437	02780437.cnv, IBM00278.ucs
278, 1143	850, 858	02780850.cnv, IBM00278.ucs
278, 1143	1051	02781051.cnv, IBM00278.ucs
278, 1143	1252, 5348	02781252.cnv, IBM00278.ucs
278, 1143	1275	02781275.cnv, IBM00278.ucs
278, 1143	1200, 1208, 13488, 17584	IBM00278.ucs
280, 1144	437	02800437.cnv, IBM00280.ucs
280, 1144	850, 858	02800850.cnv, IBM00280.ucs
280, 1144	1051	02801051.cnv, IBM00280.ucs
280, 1144	1252, 5348	02801252.cnv, IBM00280.ucs
280, 1144	1275	02801275.cnv, IBM00280.ucs
280, 1144	1200, 1208, 13488, 17584	IBM00280.ucs
284, 1145	437	02840437.cnv, IBM00284.ucs
284, 1145	850, 858	02840850.cnv, IBM00284.ucs
284, 1145	1051	02841051.cnv, IBM00284.ucs
284, 1145	1252, 5348	02841252.cnv, IBM00284.ucs
284, 1145	1275	02841275.cnv, IBM00284.ucs
284, 1145	1200, 1208, 13488, 17584	IBM00284.ucs
285, 1146	437	02850437.cnv, IBM00285.ucs

Database server CCSIDs/CPGIDs	**Database client CCSIDs/CPGIDs**	**Conversion table files**
285, 1146	850, 858	02850850.cnv, IBM00285.ucs
285, 1146	1051	02851051.cnv, IBM00285.ucs
285, 1146	1252, 5348	02851252.cnv, IBM00285.ucs
285, 1146	1275	02851275.cnv, IBM00285.ucs
285, 1146	1200, 1208, 13488, 17584	IBM00285.ucs
297, 1147	437	02970437.cnv, IBM00297.ucs
297, 1147	850, 858	02970850.cnv, IBM00297.ucs
297, 1147	1051	02971051.cnv, IBM00297.ucs
297, 1147	1252, 5348	02971252.cnv, IBM00297.ucs
297, 1147	1275	02971275.cnv, IBM00297.ucs
297, 1147	1200, 1208, 13488, 17584	IBM00297.ucs
437	850, 858	04370850.cnv, 08500437.cnv
500, 1148	437	05000437.cnv, IBM00500.ucs
500, 1148	850, 858	05000850.cnv, IBM00500.ucs
500, 1148	857, 9049	05000857.cnv, IBM00500.ucs
500, 1148	920	05000920.cnv, IBM00500.ucs
500, 1148	1051	05001051.cnv, IBM00500.ucs
500, 1148	1114, 5210	05001114.cnv, IBM00500.ucs
500, 1148	1252, 5348	05001252.cnv, IBM00500.ucs
500, 1148	1254, 5350	05001254.cnv, IBM00500.ucs
500, 1148	1275	05001275.cnv, IBM00500.ucs
500, 1148	1200, 1208, 13488, 17584	IBM00500.ucs
850, 858	437	08500437.cnv, 04370850.cnv
850, 858	860	08500860.cnv, 08600850.cnv
850, 858	1114, 5210	08501114.cnv, 11140850.cnv
850, 858	1275	08501275.cnv, 12750850.cnv
860	850, 858	08600850.cnv, 08500860.cnv
871, 1149	437	08710437.cnv, IBM00871.ucs
871, 1149	850, 858	08710850.cnv, IBM00871.ucs
871, 1149	1051	08711051.cnv, IBM00871.ucs
871, 1149	1252, 5348	08711252.cnv, IBM00871.ucs
871, 1149	1275	08711275.cnv, IBM00871.ucs

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
871, 1149	1200, 1208, 13488, 17584	IBM00871.ucs
1275	850, 858	12750850.cnv, 08501275.cnv

Latin-2:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
852, 9044	1250, 5346	08521250.cnv, 12500852.cnv
870, 1153	852, 9044	08700852.cnv, IBM00870.ucs
870, 1153	1250, 5346	08701250.cnv, IBM00870.ucs
870, 1153	1200, 1208, 13488, 17584	IBM00870.ucs
1250, 5346	852, 9044	12500852.cnv, 08521250.cnv, IBM01250.ucs
1250, 5346	1200, 1208, 13488, 17584	IBM01250.ucs

Simplified Chinese:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
837, 935, 1388	1200, 1208, 13488, 17584	1388ucs2.cnv
1386	1200, 1208, 13488, 17584	1386ucs2.cnv, ucs21386.cnv

Traditional Chinese:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
937, 835, 1371	950, 1370	09370950.cnv, 0937ucs2.cnv
937, 835, 1371	1200, 1208, 13488, 17584	0937ucs2.cnv
1114, 5210	850, 858	11140850.cnv, 08501114.cnv

Thailand:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
838, 1160	874, 1161	08380874.cnv, IBM00838.ucs
838, 1160	1200, 1208, 13488, 17584	IBM00838.ucs

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
874, 1161	1200, 1208, 13488, 17584	IBM00874.ucs

Turkish:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
857, 9049	1254, 5350	08571254.cnv, 12540857.cnv
1026, 1155	857, 9049	10260857.cnv, IBM01026.ucs
1026, 1155	1254, 5350	10261254.cnv, IBM01026.ucs
1026, 1155	1200, 1208, 13488, 17584	IBM01026.ucs
1254, 5350	857, 9049	12540857.cnv, 08571254.cnv, IBM01254.ucs
1254, 5350	1200, 1208, 13488, 17584	IBM01254.ucs

Ukraine:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
1123, 1158	1124	11231124.cnv
1123, 1158	1125, 848	11231125.cnv
1123, 1158	1251, 5347	11231251.cnv
1124	1251, 5347	11241251.cnv, 12511124.cnv
1125, 848	1251, 5347	11251251.cnv, 12511125.cnv

Unicode:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
1200, 1208, 13488, 17584	813, 4909	IBM00813.ucs
1200, 1208, 13488, 17584	862, 867	IBM00862.ucs
1200, 1208, 13488, 17584	864, 17248	IBM00864.ucs
1200, 1208, 13488, 17584	874, 1161	IBM00874.ucs

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
1200, 1208, 13488, 17584	921, 901	IBM00921.ucs
1200, 1208, 13488, 17584	922, 902	IBM00922.ucs
1200, 1208, 13488, 17584	954	ucs20954.cnv, 0954ucs2.cnv
1200, 1208, 13488, 17584	1046, 9238	IBM01046.ucs
1200, 1208, 13488, 17584	1250, 5346	IBM01250.ucs
1200, 1208, 13488, 17584	1251, 5347	IBM01251.ucs
1200, 1208, 13488, 17584	1253, 5349	IBM01253.ucs
1200, 1208, 13488, 17584	1254, 5350	IBM01254.ucs
1200, 1208, 13488, 17584	1255, 5351	IBM01255.ucs
1200, 1208, 13488, 17584	1256, 5352	IBM01256.ucs
1200, 1208, 13488, 17584	1386	ucs21386.cnv, 1386ucs2.cnv

Vietnamese:

Database server CCSIDs/CPGIDs	Database client CCSIDs/CPGIDs	Conversion table files
1130, 1164	1258, 5354	11301258.cnv
1258, 5354	1129, 1163	12581129.cnv

Related concepts:

- "Character conversion" in the *SQL Reference, Volume 1*

Related tasks:

- "Enabling and disabling euro symbol support" on page 245

Conversion tables for code pages 923 and 924

The following is a list of all the code page conversion table files that are associated with code pages 923 and 924. Each file is of the form XXXXYYYY.cnv or ibmZZZZZ.ucs, where XXXXX is the source code page number and YYYY is the target code page number. The file ibmZZZZZ.ucs supports conversion between code page ZZZZZ and Unicode.

To activate a particular code page conversion table, rename or copy that conversion table file to its new name as shown in the second column.

For example, to support the euro symbol when connecting a 8859-1/15 (Latin 1/9) client to a Windows 1252 database, you need to rename or copy the following code page conversion table files in the sqllib/conv/ directory:

- 09231252.cnv to 08191252.cnv
- 12520923.cnv to 12520819.cnv
- ibm00923.ucs to ibm00819.ucs

923 and 924 conversion table files in the sqlllib/conv/ directory	**New name**
00370923.cnv	00370819.cnv
02730923.cnv	02730819.cnv
02770923.cnv	02770819.cnv
02780923.cnv	02780819.cnv
02800923.cnv	02800819.cnv
02840923.cnv	02840819.cnv
02850923.cnv	02850819.cnv
02970923.cnv	02970819.cnv
04370923.cnv	04370819.cnv
05000923.cnv	05000819.cnv
08500923.cnv	08500819.cnv
08600923.cnv	08600819.cnv
08630923.cnv	08630819.cnv
08710923.cnv	08710819.cnv
09230437.cnv	08190437.cnv
09230500.cnv	08190500.cnv
09230850.cnv	08190850.cnv
09230860.cnv	08190860.cnv

923 and 924 conversion table files in the sqlllib/conv/ directory	New name
09230863.cnv	08190863.cnv
09231043.cnv	08191043.cnv
09231051.cnv	08191051.cnv
09231114.cnv	08191114.cnv
09231252.cnv	08191252.cnv
09231275.cnv	08191275.cnv
09241252.cnv	10471252.cnv
10430923.cnv	10430819.cnv
10510923.cnv	10510819.cnv
11140923.cnv	11140819.cnv
12520923.cnv	12520819.cnv
12750923.cnv	12750819.cnv
ibm00923.ucs	ibm00819.ucs

Related concepts:

- "Character conversion" in the *SQL Reference, Volume 1*

Related tasks:

- "Enabling and disabling euro symbol support" on page 245

Choosing a language for your database

When you create a database, you have to decide what language your data will be stored in. When you create a database, you can specify the territory and code set. The territory and code set may be different from the current operating system settings. If you do not explicitly choose a territory and code set at database creation time, the database will be created using the current locale. When you are choosing a code set, make sure it can encode all the characters in the language you will be using.

Another option is to store data in a Unicode database, which means that you do not have to choose a specific language; Unicode encoding includes characters from almost all of the living languages in the world.

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.

Enabling bidirectional support

Bidirectional layout transformations are implemented in DB2 Universal Database using the new Coded Character Set Identifier (CCSID) definitions. For the new bidirectional-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.

Restrictions:

The following restrictions apply:

- 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.
- CCSID override is not supported for cases where the HOST EBCDIC platform is the client, and DB2 UDB is the server.

Procedure:

To specify a particular bidirectional CCSID in a non-DRDA environment:

- Ensure the DB2BIDI registry variable is set to YES.
- Select the CCSID 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.

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.

Related concepts:

- "Bidirectional support with DB2 Connect" on page 262
- "Handling BiDi data" in the *DB2 Connect User's Guide*

Related reference:

- "Bidirectional-specific CCSIDs" on page 259
- "General registry variables" in the *Administration Guide: Performance*

Bidirectional-specific CCSIDs

The following bidirectional attributes are required for correct handling of bidirectional data on different platforms:

- Text type
- Numeric shaping
- Orientation
- Text shaping
- Symmetric swapping

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 z/OS and OS/390 usually use SHAPED VISUAL data. Therefore, without support for bidirectional attributes, data sent from DB2 Universal Database for OS/390 and z/OS to DB2 UDB on Windows 32-bit operating systems may display incorrectly.

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 as shown in Table 24 on page 260. CDRA string types are defined as shown in Table 25 on page 261.

Table 24. Bidirectional CCSIDs

CCSID	Code Page	String Type
420	420	4
424	424	4
856	856	5
862	862	4
864	864	5
867	862	4
916	916	5
1046	1046	5
1089	1089	5
1255	1255	5
1256	1256	5
5351	1255	5
5352	1256	5
8612	420	5
8616	424	10
9048	856	5
9238	1046	5
12712	424	4
16804	420	4
17248	864	5
62208	856	4
62209	862	10
62210	916	4
62211	424	5
62213	862	5
62215	1255	4
62218	864	4
62220	856	6
62221	862	6
62222	916	6
62223	1255	6
62224	420	6

Table 24. Bidirectional CCSIDs (continued)

CCSID	Code Page	String Type
62225	864	6
62226	1046	6
62227	1089	6
62228	1256	6
62229	424	8
62230	856	8
62231	862	8
62232	916	8
62233	420	8
62234	420	9
62235	424	6
62236	856	10
62237	1255	8
62238	916	10
62239	1255	10
62240	424	11
62241	856	11
62242	862	11
62243	916	11
62244	1255	11
62245	424	10
62246	1046	8
62247	1046	9
62248	1046	4
62249	1046	12
62250	420	12

Table 25. CDRA string types

String type	Text type	Numeric shaping	Orientation	Text shaping	Symmetrical swapping
4	Visual	Passthrough	LTR	Shaped	Off
5	Implicit	Arabic	LTR	Unshaped	On
6	Implicit	Arabic	RTL	Unshaped	On

Table 25. CDRA string types (continued)

String type	Text type	Numeric shaping	Orientation	Text shaping	Symmetrical swapping
7*	Visual	Passthrough	Contextual*	Unshaped ligature	Off
8	Visual	Passthrough	RTL	Shaped	Off
9	Visual	Passthrough	RTL	Shaped	On
10	Implicit	Arabic	Contextual LTR	Unshaped	On
11	Implicit	Arabic	Contextual RTL	Unshaped	On
12	Implicit	Arabic	RTL	Shaped	Off

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.

Related concepts:

- "Bidirectional support with DB2 Connect" on page 262

Related tasks:

- "Enabling bidirectional support" on page 258

Bidirectional support with DB2 Connect

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 or iSeries™ 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 or iSeries 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.

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 or iSeries database running CCSID 00424 (bidirectional string type 4). However, you know that the data contained in the DB2 host or iSeries database is based on CCSID 08616 (bidirectional string type 6).

There are two problems here: The first is that the DB2 host or iSeries database does not know the difference in the bidirectional string types with CCSIDs 00424 and 08616. The second problem is that the DB2 host or iSeries 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 or iSeries 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 or iSeries database. You will need to use following catalog command for the DB2 host or iSeries database:

```
db2 catalog dcs database nydb1 as telaviv parms ",,,,,,,,BIDI=08616"
```

This command tells DB2 Connect to override the DB2 host or iSeries database CCSID of 00424 with 08616. This override includes the following processing:

1. DB2 Connect connects to the DB2 host or iSeries database using CCSID 00862.
2. DB2 Connect performs bidirectional layout transformation on the data it is about to *send* to the DB2 host or iSeries 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 or iSeries 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.

Related concepts:

- "Handling BiDi data" in the *DB2 Connect User's Guide*

Related reference:

- "Bidirectional-specific CCSIDs" on page 259

Collating Sequences

The database manager compares character data using a *collating sequence*. This is an ordering for a set of characters that determines whether a particular character sorts higher, lower, or the same as another.

Note: Character string data defined with the FOR BIT DATA attribute, and BLOB data, is sorted using the binary sort sequence.

For example, a collating sequence can be used to indicate that lowercase and uppercase versions of a particular character are to be sorted equally.

The database manager allows databases to be created with custom collating sequences. The following sections help you determine and implement a particular collating sequence for a database.

Each single-byte character in a database is represented internally as a unique number between 0 and 255 (in hexadecimal notation, between X'00' and X'FF'). This number is referred to as the *code point* of the character; the assignment of numbers to characters in a set is collectively called a *code page*. A collating sequence is a mapping between the code point and the desired position of each character in a sorted sequence. The numeric value of the position is called the *weight* of the character in the collating sequence. In the simplest collating sequence, the weights are identical to the code points. This is called the *identity sequence*.

For example, suppose the characters B and b have the code points X'42' and X'62', respectively. If (according to the collating sequence table) they both have a sort weight of X'42' (B), they collate the same. If the sort weight for B is X'9E', and the sort weight for b is X'9D', b will be sorted before B. The collation sequence table specifies the weight of each character. The table is different from a code page, which specifies the code point of each character.

Consider the following example. The ASCII characters A through Z are represented by X'41' through X'5A'. To describe a collating sequence in which these characters are sorted consecutively (no intervening characters), you can write: X'41', X'42', ... X'59', X'5A'.

The hexadecimal value of a multi-byte character is also used as the weight. For example, suppose the code points for the double-byte characters A and B are X'8260' and X'8261' respectively, then the collation weights for X'82', X'60', and X'61' are used to sort these two characters according to their code points.

The weights in a collating sequence need not be unique. For example, you could give uppercase letters and their lowercase equivalents the same weight.

Specifying a collating sequence can be simplified if the collating sequence provides weights for all 256 code points. The weight of each character can be determined using the code point of the character.

In all cases, DB2® uses the collation table that was specified at database creation time. If you want the multi-byte characters to be sorted the way that they appear in their code point table, you must specify IDENTITY as the collation sequence when you create the database.

Note: For double-byte and Unicode characters in GRAPHIC fields, the sort sequence is always IDENTITY.

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.

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.

Related concepts:
- "Character conversion" in the *SQL Reference, Volume 1*
- "Character Comparisons Based on Collating Sequences" in the *Application Development Guide: Programming Client Applications*

Collating Thai characters

Thai contains special vowels ("leading vowels"), tonal marks and other special characters that are not sorted sequentially.

Restrictions:

You must create your database with a Thai locale and code set.

Procedure:

When you create the database, use the COLLATE USING NLSCHAR clause on the CREATE DATABASE command.

Related concepts:
- "Collating Sequences" on page 264

Related reference:
- "CREATE DATABASE Command" in the *Command Reference*

Date and time formats by territory code

The character string representation of date and time formats is the default format of datetime values associated with the territory code of the application. This default format can be overridden by specifying the DATETIME 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 territory 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 26.
- Output Date Format
  - The default output date format is shown in Table 26.

    Note: Table 26 also shows a listing of the string formats for the various territory codes.

Table 26. Date and Time Formats by Territory Code

Territory 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

Table 26. Date and Time Formats by Territory Code (continued)

Territory Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
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
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

Table 26. Date and Time Formats by Territory Code (continued)

Territory Code	Local Date Format	Local Time Format	Default Output Date Format	Input Date Formats
042 Slovakia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
386 Slovenia	yyyy-mm-dd	JIS	ISO	LOC, USA, EUR, ISO
034 Spain	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
046 Sweden	dd/mm/yyyy	ISO	ISO	LOC, EUR, ISO
041 Switzerland	dd/mm/yyyy	ISO	EUR	LOC, EUR, ISO
088 Taiwan	mm-dd-yyyy	JIS	ISO	LOC, USA, EUR, ISO
066 Thailand (2)	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
090 Turkey	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
044 UK	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO
001 USA	mm-dd-yyyy	JIS	USA	LOC, USA, EUR, ISO
084 Vietnam	dd/mm/yyyy	JIS	LOC	LOC, EUR, ISO

Notes:

1. Countries/Regions using the default C locale are assigned territory code 001.
2. yyyy in Buddhist era is equivalent to Gregorian + 543 years (Thailand only).

Related reference:

- "BIND Command" in the *Command Reference*
- "PRECOMPILE Command" in the *Command Reference*

Unicode character encoding

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.

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).

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 or supplementary 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.

UCS-2

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. UCS-2 can encode all the (16-bit) characters defined in the Unicode version 3.0 repertoire. Two UCS-2 characters — a high followed by a low surrogate — are required to encode each of the new supplementary characters introduced starting in Unicode version 3.1. These supplementary characters are defined outside the original 16-bit Basic Multilingual Plane (BMP or Plane 0).

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. UTF-8 can encode both non-supplementary and supplementary characters.

UTF-16

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.

The number of bytes for each UTF-16 character in UTF-8 format can be determined from Table 27.

Table 27. 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.

Related concepts:

- "Unicode implementation in DB2" on page 271
- "Unicode handling of data types" on page 274
- "Unicode literals" on page 276

Related tasks:

- "Creating a Unicode database" on page 275

Unicode implementation in DB2

DB2® UDB supports UTF-8 and UCS-2, that is, Unicode without surrogates.

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.

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. 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. In other usages, DB2 continues to treat a surrogate pair as two independent UCS-2 characters. You can safely store supplementary characters in DB2 Unicode databases, provided you know how to distinguish them from the non-supplementary characters.

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).

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 non-supplementary Unicode characters, their binary collation orders when encoded in UTF-8 and UCS-2 are the same. But if you have any supplementary 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, namely X'EFBFBF'. But in UCS-2, the same supplementary character is encoded as a pair of UCS-2 high and low surrogate characters, and has the binary form of 1101 1000 xxxx xxxx 1101 1100 xxxx xxxx, which is less than the UCS-2 encoding of U+FFFF.

A Unicode database can also be created with the IDENTITY_16BIT collation option. IDENTITY_16BIT differs from the default IDENTITY collation option in that the CHAR, VARCHAR, LONG VARCHAR, and CLOB data in the Unicode database will be collated using the CESU-8 binary order instead of the UTF-8 binary order. CESU-8 is the *Compatibility Encoding Scheme for UTF-16: 8-Bit*, and as of this writing, its specification is contained in the Draft Unicode Technical Report #26 available at the Unicode Technical Consortium web site (www.unicode.org). CESU-8 is binary identical to UTF-8 except for the Unicode supplementary characters, that is, those characters that are defined outside the 16-bit Basic Multilingual Plane (BMP or Plane 0). In UTF-8 encoding, a supplementary character is represented by one four-byte sequence, but the same character in CESU-8 requires two three-byte sequences. Using the IDENTITY_16BIT collation option will yield the same collation order for both character and graphic data. Note that CHAR, VARCHAR, LONG VARCHAR, and CLOB data continue to be stored in UTF-8 format even when the IDENTITY_16BIT option is specified.

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.

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.

Related concepts:

- "Unicode character encoding" on page 269
- "Unicode handling of data types" on page 274
- "Unicode literals" on page 276

Related tasks:

- "Creating a Unicode database" on page 275

Unicode handling of 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 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.

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.

Related concepts:
- “Unicode character encoding” on page 269
- “Unicode implementation in DB2” on page 271

Creating a Unicode database

Procedure:

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.

To create a Unicode database with the territory code for the United States of America:

```
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 *sqledbterritoryinfo* accordingly. For example, set SQLDBCODESET to `UTF-8`, and SQLDBLOCALE to any valid territory code (for example, `US`).

Related concepts:

- "Unicode implementation in DB2" on page 271

Related reference:

- "sqlecrea - Create Database" in the *Administrative API Reference*
- "CREATE DATABASE Command" in the *Command Reference*

Unicode literals

Unicode literals can be specified in two ways:

- As a graphic string constant, using the G'...' or N'....' format. 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).

Related concepts:

- "Unicode character encoding" on page 269
- "Unicode implementation in DB2" on page 271

Related reference:

- "Constants" in the *SQL Reference, Volume 1*

String comparisons in a Unicode 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. You need two underscores to match one extended GRAPHIC character because one extended GRAPHIC character is represented by two UCS-2 characters in a GRAPHIC column.

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.

Related concepts:
- "Unicode character encoding" on page 269
- "Unicode implementation in DB2" on page 271

Related reference:
- "Character strings" in the *SQL Reference, Volume 1*
- "Graphic strings" in the *SQL Reference, Volume 1*

Appendix C. DB2 Universal Database technical information

Overview of DB2 Universal Database technical information

DB2 Universal Database technical information can be obtained in the following formats:

- Books (PDF and hard-copy formats)
- A topic tree (HTML format)
- Help for DB2 tools (HTML format)
- Sample programs (HTML format)
- Command line help
- Tutorials

This section is an overview of the technical information that is provided and how you can access it.

FixPaks for DB2 documentation

IBM may periodically make documentation FixPaks available. Documentation FixPaks allow you to update the information that you installed from the *DB2 HTML Documentation CD* as new information becomes available.

Note: If you do install documentation FixPaks, your HTML documentation will contain more recent information than either the DB2 printed or online PDF manuals.

Categories of DB2 technical information

The DB2 technical information is categorized by the following headings:

- Core DB2 information
- Administration information
- Application development information
- Business intelligence information
- DB2 Connect information
- Getting started information
- Tutorial information
- Optional component information
- Release notes

The following tables describe, for each book in the DB2 library, the information needed to order the hard copy, print or view the PDF, or locate the HTML directory for that book. A full description of each of the books in the DB2 library is available from the IBM Publications Center at www.ibm.com/shop/publications/order

The installation directory for the HTML documentation CD differs for each category of information:

```
htmlcdpath/doc/htmlcd/%L/category
```

where:

- *htmlcdpath* is the directory where the HTML CD is installed.
- *%L* is the language identifier. For example, `en_US`.
- *category* is the category identifier. For example, `core` for the core DB2 information.

In the PDF file name column in the following tables, the character in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d1e80` identifies the English version of the *Administration Guide: Planning* and the file name `db2d1g80` identifies the German 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**
Arabic	w
Brazilian Portuguese	b
Bulgarian	u
Croatian	9
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
Romanian	8
Russian	r
Simp. Chinese	c
Slovakian	7
Slovenian	l

Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

No form number indicates that the book is only available online and does not have a printed version.

Core DB2 information

The information in this category covers DB2 topics that are fundamental to all DB2 users. You will find the information in this category useful whether you are a programmer, a database administrator, or you work with DB2 Connect, DB2 Warehouse Manager, or other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/core`.

Table 28. Core DB2 information

Name	Form Number	PDF File Name
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
IBM DB2 Universal Database Glossary	No form number	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Message Reference, Volume 1	GC09-4840	db2m1x80
IBM DB2 Universal Database Message Reference, Volume 2	GC09-4841	db2m2x80
IBM DB2 Universal Database What's New	SC09-4848	db2q0x80

Administration information

The information in this category covers those topics required to effectively design, implement, and maintain DB2 databases, data warehouses, and federated systems.

The installation directory for this category is `doc/htmlcd/%L/admin`.

Table 29. Administration information

Name	Form number	PDF file name
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guide to GUI Tools for Administration and Development	SC09-4851	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Installing and Administering a Satellite Environment	GC09-4823	db2dsx80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Application development information

The information in this category is of special interest to application developers or programmers working with DB2. You will find information about supported languages and compilers, as well as the documentation required to access DB2 using the various supported programming interfaces, such as embedded SQL, ODBC, JDBC, SQLj, and CLI. If you view this information online in HTML you can also access a set of DB2 sample programs in HTML.

The installation directory for this category is `doc/htmlcd/%L/ad`.

Table 30. Application development information

Name	Form number	PDF file name
IBM DB2 Universal Database Application Development Guide: Building and Running Applications	SC09-4825	db2axx80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

Business intelligence information

The information in this category describes how to use components that enhance the data warehousing and analytical capabilities of DB2 Universal Database.

The installation directory for this category is `doc/htmlcd/%L/wareh`.

Table 31. Business intelligence information

Name	Form number	PDF file name
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Installation Guide	GC27-1122	db2idx80

DB2 Connect information

The information in this category describes how to access host or iSeries data using DB2 Connect Enterprise Edition or DB2 Connect Personal Edition.

The installation directory for this category is `doc/htmlcd/%L/conn`.

Table 32. DB2 Connect information

Name	Form number	PDF file name
APPC, CPI-C, and SNA Sense Codes	No form number	db2apx80
IBM Connectivity Supplement	No form number	db2h1x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition	GC09-4833	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Getting started information

The information in this category is useful when you are installing and configuring servers, clients, and other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/start`.

Table 33. Getting started information

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Clients	GC09-4832	db2itx80

Table 33. Getting started information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Servers	GC09-4836	db2isx80
IBM DB2 Universal Database Quick Beginnings for DB2 Personal Edition	GC09-4838	db2i1x80
IBM DB2 Universal Database Installation and Configuration Supplement	GC09-4837	db2iyx80
IBM DB2 Universal Database Quick Beginnings for DB2 Data Links Manager	GC09-4829	db2z6x80

Tutorial information

Tutorial information introduces DB2 features and teaches how to perform various tasks.

The installation directory for this category is `doc/htmlcd/%L/tutr`.

Table 34. Tutorial information

Name	Form number	PDF file name
Business Intelligence Tutorial: Introduction to the Data Warehouse	No form number	db2tux80
Business Intelligence Tutorial: Extended Lessons in Data Warehousing	No form number	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	No form number	db2tdx80
Information Catalog Center Tutorial	No form number	db2aix80
Video Central for e-business Tutorial	No form number	db2twx80
Visual Explain Tutorial	No form number	db2tvx80

Optional component information

The information in this category describes how to work with optional DB2 components.

The installation directory for this category is `doc/htmlcd/%L/opt`.

Table 35. Optional component information

Name	Form number	PDF file name
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6740	N/A

Release notes

The release notes provide additional information specific to your product's release and FixPak level. They also provides summaries of the documentation updates incorporated in each release and FixPak.

Table 36. Release notes

Name	Form number	PDF file name	HTML directory
DB2 Release Notes	See note.	See note.	See note.
DB2 Installation Notes	Available on product CD-ROM only.	Available on product CD-ROM only.	

Note: The HTML version of the release notes 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/opt/db2_08_01` on AIX
- `/opt/IBM/db2/V8.1` on all other UNIX operating systems

Related tasks:

- "Printing DB2 books from PDF files" on page 287
- "Ordering printed DB2 books" on page 288
- "Accessing online help" on page 288
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 292
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 294

Printing DB2 books from PDF files

You can print DB2 books from the PDF files on the *DB2 PDF Documentation* CD. Using Adobe Acrobat Reader, you can print either the entire book or a specific range of pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com

Procedure:

To print a DB2 book from a PDF file:

1. Insert the *DB2 PDF Documentation* CD. On UNIX operating systems, mount the DB2 PDF Documentation CD. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
   - On Windows operating systems:

     `x:\doc\language` directory, where *x* represents the CD-ROM drive letter and *language* represents the two-character territory code that represents your language (for example, EN for English).
   - On UNIX operating systems:

     `/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.

Related tasks:

- "Ordering printed DB2 books" on page 288
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 292
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 294

Related reference:
- "Overview of DB2 Universal Database technical information" on page 279

Ordering printed DB2 books

Procedure:

To order printed books:
- Contact your IBM authorized dealer or marketing representative. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/planetwide
- Phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada.
- Visit the IBM Publications Center at www.ibm.com/shop/publications/order

You can also obtain printed DB2 manuals by ordering Doc Packs for your DB2 product from your IBM Reseller. The Doc Packs are subsets of the manuals in the DB2 library selected to help you to get started using the DB2 product that you purchased. The manuals in the Doc Packs are the same as those that are available in PDF format on the *DB2 PDF Documentation CD* and contain the same content as the documentation that is available on the *DB2 HTML Documentation CD*.

Related tasks:
- "Printing DB2 books from PDF files" on page 287
- "Finding topics by accessing the DB2 Information Center from a browser" on page 290
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 294

Related reference:
- "Overview of DB2 Universal Database technical information" on page 279

Accessing online help

The online help that comes with all DB2 components is available in three types:
- Window and notebook help
- Command line help
- SQL statement help

Window and notebook help explain the tasks that you can perform in a window or notebook and describe the controls. This help has two types:
- Help accessible from the **Help** button
- Infopops

The **Help** button gives you access to overview and prerequisite information. The infopops describe the controls in the window or notebook. Window and notebook help are available from DB2 centers and components that have user interfaces.

Command line help includes Command help and Message help. Command help explains the syntax of commands in the command line processor. Message help describes the cause of an error message and describes any action you should take in response to the error.

SQL statement help includes SQL help and SQLSTATE help. DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the syntax of SQL statements (SQL states and class codes).

Note: SQL help is not available for UNIX operating systems.

Procedure:

To access online help:
- For window and notebook help, click **Help** or click that control, then click **F1**. If the **Automatically display infopops** check box on the **General** page of the **Tool Settings** notebook is selected, you can also see the infopop for a particular control by holding the mouse cursor over the control.
- For command line help, open the command line processor and enter:
  - For Command help:
    ```
    ? 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.
- For Message help:
  ```
  ? XXXnnnnn
  ```
  where *XXXnnnnn* represents a valid message identifier.

  For example, `? SQL30081` displays help about the SQL30081 message.

- For SQL statement help, open the command line processor and 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.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 290
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 294

Finding topics by accessing the DB2 Information Center from a browser

The DB2 Information Center accessed from a browser enables you to access the information you need to take full advantage of DB2 Universal Database and DB2 Connect. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, metadata, and DB2 extenders.

The DB2 Information Center accessed from a browser is composed of the following major elements:

Navigation tree
: The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topics, the glossary, and the master index in the DB2 Information Center.

Navigation toolbar
: The navigation toolbar is located in the top right frame of the browser window. The navigation toolbar contains buttons that enable you to search the DB2 Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame
: The content frame is located in the bottom right frame of the browser window. The content frame displays topics from the DB2 Information Center when you click on a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index.

Prerequisites:

To access the DB2 Information Center from a browser, you must use one of the following browsers:

- Microsoft Explorer, version 5 or later
- Netscape Navigator, version 6.1 or later

Restrictions:

The DB2 Information Center contains only those sets of topics that you chose to install from the *DB2 HTML Documentation CD*. If your Web browser returns a `File not found` error when you try to follow a link to a topic, you must install one or more additional sets of topics from the *DB2 HTML Documentation CD*.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter one or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field. The numerical ranking beside the hit provides an indication of the strength of the match (bigger numbers indicate stronger matches).

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon of the category of topics related to your area of interest. A list of subcategories displays underneath the icon.
2. Continue to click the book icons until you find the category containing the topics in which you are interested. Categories that link to topics display the category title as an underscored link when you move the cursor over the category title. The navigation tree identifies topics with a page icon.
3. Click the topic link. The topic displays in the content frame.

To find a topic or term in the master index:

1. In the navigation tree, click the “Index” category. The category expands to display a list of links arranged in alphabetical order in the navigation tree.
2. In the navigation tree, click the link corresponding to the first character of the term relating to the topic in which you are interested. A list of terms

with that initial character displays in the content frame. Terms that have multiple index entries are identified by a book icon.

3. Click the book icon corresponding to the term in which you are interested. A list of subterms and topics displays below the term you clicked. Topics are identified by page icons with an underscored title.
4. Click on the title of the topic that meets your needs. The topic displays in the content frame.

Related concepts:

- "Accessibility" on page 299
- "DB2 Information Center accessed from a browser" on page 302

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 292
- "Updating the HTML documentation installed on your machine" on page 294
- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 297
- "Searching the DB2 documentation" on page 298

Related reference:

- "Overview of DB2 Universal Database technical information" on page 279

Finding product information by accessing the DB2 Information Center from the administration tools

The DB2 Information Center provides quick access to DB2 product information and is available on all operating systems for which the DB2 administration tools are available.

The DB2 Information Center accessed from the tools provides six types of information.

Tasks Key tasks you can perform using DB2.

Concepts
Key concepts for DB2.

Reference
DB2 reference information, such as keywords, commands, and APIs.

Troubleshooting
Error messages and information to help you with common DB2 problems.

Samples
Links to HTML listings of the sample programs provided with DB2.

Tutorials
Instructional aid designed to help you learn a DB2 feature.

Prerequisites:

Some links in the DB2 Information Center point to Web sites on the Internet. To display the content for these links, you will first have to connect to the Internet.

Procedure:

To find product information by accessing the DB2 Information Center from the tools:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Click the tab of the information type related to the information you are attempting to find.
3. Navigate through the tree and click on the topic in which you are interested. The Information Center will then launch a Web browser to display the information.
4. To find information without browsing the lists, click the **Search** icon to the right of the list.

   Once the Information Center has launched a browser to display the information, you can perform a full-text search by clicking the **Search** icon in the navigation toolbar.

Related concepts:

- "Accessibility" on page 299
- "DB2 Information Center accessed from a browser" on page 302

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 290
- "Searching the DB2 documentation" on page 298

Viewing technical documentation online directly from the DB2 HTML Documentation CD

All of the HTML topics that you can install from the *DB2 HTML Documentation CD* can also be read directly from the CD. Therefore, you can view the documentation without having to install it.

Restrictions:

As the Tools help is installed from the DB2 product CD and not from the *DB2 HTML Documentation CD*, you must install the DB2 product to view the help.

Procedure:

1. Insert the *DB2 HTML Documentation* CD. On UNIX operating systems, mount the *DB2 HTML Documentation CD*. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start your HTML browser and open the appropriate file:
   - For Windows operating systems:
     ```
     e:\program files\IBM\SQLLIB\doc\htmlcd\%L\index.htm
     ```
     where *e* represents the CD-ROM drive, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.
   - For UNIX operating systems:
     ```
     /cdrom/program files/IBM/SQLLIB/doc/htmlcd/%L/index.htm
     ```
     where */cdrom/* represents where the CD is mounted, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 290
- "Copying files from the DB2 HTML Documentation CD to a Web server" on page 296

Related reference:

- "Overview of DB2 Universal Database technical information" on page 279

Updating the HTML documentation installed on your machine

It is now possible to update the HTML installed from the *DB2 HTML Documentation CD* when updates are made available from IBM. This can be done in one of two ways:

- Using the Information Center (if you have the DB2 administration GUI tools installed).
- By downloading and applying a DB2 HTML documentation FixPak .

Note: This will NOT update the DB2 code; it will only update the HTML documentation installed from the *DB2 HTML Documentation CD*.

Procedure:

To use the Information Center to update your local documentation:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Ensure your machine has access to the external Internet; the updater will download the latest documentation FixPak from the IBM server if required.
3. Select **Information Center —> Update Local Documentation** from the menu to start the update.
4. Supply your proxy information (if required) to connect to the external Internet.

The Information Center will download and apply the latest documentation FixPak, if one is available.

To manually download and apply the documentation FixPak :

1. Ensure your machine is connected to the Internet.
2. Open the DB2 support page in your Web browser at: www.ibm.com/software/data/db2/udb/winos2unix/support.
3. Follow the link for Version 8 and look for the ″Documentation FixPaks″ link.
4. Determine if the version of your local documentation is out of date by comparing the documentation FixPak level to the documentation level you have installed. This current documentation on your machine is at the following level: **DB2 v8.1 GA**.
5. If there is a more recent version of the documentation available then download the FixPak applicable to your operating system. There is one FixPak for all Windows platforms, and one FixPak for all UNIX platforms.
6. Apply the FixPak:
   - For Windows operating systems: The documentation FixPak is a self extracting zip file. Place the downloaded documentation FixPak in an empty directory, and run it. It will create a **setup** command which you can run to install the documentation FixPak.

- For UNIX operating systems: The documentation FixPak is a compressed tar.Z file. Uncompress and untar the file. It will create a directory named `delta_install` with a script called **installdocfix**. Run this script to install the documentation FixPak.

Related tasks:
- "Copying files from the DB2 HTML Documentation CD to a Web server" on page 296

Related reference:
- "Overview of DB2 Universal Database technical information" on page 279

Copying files from the DB2 HTML Documentation CD to a Web server

The entire DB2 information library is delivered to you on the *DB2 HTML Documentation CD* and may be installed on a Web server for easier access. Simply copy to your Web server the documentation for the languages that you want.

Note: You might encounter slow performance if you access the HTML documentation from a Web server through a low-speed connection.

Procedure:

To copy files from the *DB2 HTML Documentation CD* to a Web server, use the appropriate source path:
- For Windows operating systems:
  ```
  E:\program files\IBM\SQLLIB\doc\htmlcd\%L\*.*
  ```
  where *E* represents the CD-ROM drive and *%L* represents the language identifier.
- For UNIX operating systems:
  ```
  /cdrom/program files/IBM/sqllib/doc/htmlcd/%L/*.*
  ```
  where *cdrom* represents the mount point for the CD-ROM drive and *%L* represents the language identifier.

Related tasks:
- "Searching the DB2 documentation" on page 298

Related reference:
- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "Overview of DB2 Universal Database technical information" on page 279

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by web browsers. This task describes possible workarounds.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following workaround, especially if you see the following line in the browser Java console:

```
Cannot find class  java/security/InvalidParameterException
```

- On Windows operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `x:program files\IBM\SQLLIB\doc\htmlcd\locale\InvalidParameterException.class` file to the `java\classes\java\security\` directory relative to your Netscape browser installation, where *x* represents the CD-ROM drive letter and *locale* represents the name of the desired locale.

  Note: You may have to create the `java\security\` subdirectory structure.
- On UNIX operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `/cdrom/program files/IBM/SQLLIB/doc/htmlcd/locale/InvalidParameterException.class` file to the `java/classes/java/security/` directory relative to your Netscape browser installation, where *cdrom* represents the mount point of the CD-ROM and *locale* represents the name of the desired locale.

  Note: You may have to create the `java/security/` subdirectory structure.

If your Netscape browser still fails to display the search input window, try the following:

- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the machine. Then open a new instance of the Netscape browser and try to start the search again.
- Purge the browser's cache.
- Try a different version of Netscape, or a different browser.

Related tasks:

- "Searching the DB2 documentation" on page 298

Searching the DB2 documentation

You can search the library of DB2 documentation to locate information that you need. A pop-up search window opens when you click the search icon in the navigation toolbar of the DB2 Information Center (accessed from a browser). The search can take a minute to load, depending on the speed of your computer and network.

Prerequisites:

You need Netscape 6.1 or higher, or Microsoft's Internet Explorer 5 or higher. Ensure that your browser's Java support is enabled.

Restrictions:

The following restrictions apply when you use the documentation search:

- Search is not case sensitive.
- Boolean searches are not supported.
- Wildcard and partial searches are not supported. A search on *java** (or *java*) will only look for the literal string *java** (or *java*) and would not, for example, find *javadoc*.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click the **Search** icon.
2. In the top text entry field of the Search window, enter one or more terms (separated by a space) related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field. The numerical ranking beside the hit provides an indication of the strength of the match (bigger numbers indicate stronger matches).

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** list, click the title of the topic you want to read. The topic displays in the content frame of the DB2 Information Center.

Note: When you perform a search, the first (highest-ranking) result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in the results list.

Related tasks:

- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 297

Online DB2 troubleshooting information

With the release of DB2® UDB Version 8, there will no longer be a *Troubleshooting Guide*. The troubleshooting information once contained in this guide has been integrated into the DB2 publications. By doing this, we are able to deliver the most up-to-date information possible. To find information on the troubleshooting utilities and functions of DB2, access the DB2 Information Center from any of the tools.

Refer to the DB2 Online Support site if you are experiencing problems and want help finding possible causes and solutions. The support site contains a large, constantly updated database of DB2 publications, TechNotes, APAR (product problem) records, FixPaks, and other resources. You can use the support site to search through this knowledge base and find possible solutions to your problems.

Access the Online Support site at www.ibm.com/software/data/db2/udb/winos2unix/support, or by clicking the **Online Support** button in the DB2 Information Center. Frequently changing information, such as the listing of internal DB2 error codes, is now also available from this site.

Related concepts:

- "DB2 Information Center accessed from a browser" on page 302

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 292

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in DB2® Universal Database Version 8:

- DB2 allows you to operate all features using the keyboard instead of the mouse. See "Keyboard Input and Navigation" on page 300.
- DB2 enables you customize the size and color of your fonts. See "Accessible Display" on page 300.
- DB2 allows you to receive either visual or audio alert cues. See "Alternative Alert Cues" on page 300.
- DB2 supports accessibility applications that use the Java™ Accessibility API. See "Compatibility with Assistive Technologies" on page 300.

- DB2 comes with documentation that is provided in an accessible format. See “Accessible Documentation”.

Keyboard Input and Navigation

Keyboard Input

You can operate the DB2 Tools using only the keyboard. You can use keys or key combinations to perform most operations that can also be done using a mouse.

Keyboard Focus

In UNIX-based systems, the position of the keyboard focus is highlighted, indicating which area of the window is active and where your keystrokes will have an effect.

Accessible Display

The DB2 Tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font Settings

The DB2 Tools allow you to select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

Non-dependence on Color

You do not need to distinguish between colors in order to use any of the functions in this product.

Alternative Alert Cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with Assistive Technologies

The DB2 Tools interface supports the Java Accessibility API enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible Documentation

Documentation for the DB2 family of products is available in HTML format. This allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

DB2 tutorials

The DB2® tutorials help you learn about various aspects of DB2 Universal Database. The tutorials provide lessons with step-by-step instructions in the areas of developing applications, tuning SQL query performance, working with data warehouses, managing metadata, and developing Web services using DB2.

Before you begin:

Before you can access these tutorials using the links below, you must install the tutorials from the *DB2 HTML Documentation* CD-ROM.

If you do not want to install the tutorials, you can view the HTML versions of the tutorials directly from the *DB2 HTML Documentation CD*. PDF versions of these tutorials are also available on the *DB2 PDF Documentation CD*.

Some tutorial lessons use sample data or code. See each individual tutorial for a description of any prerequisites for its specific tasks.

DB2 Universal Database tutorials:

If you installed the tutorials from the *DB2 HTML Documentation* CD-ROM, you can click on a tutorial title in the following list to view that tutorial.

Business Intelligence Tutorial: Introduction to the Data Warehouse Center
: Perform introductory data warehousing tasks using the Data Warehouse Center.

Business Intelligence Tutorial: Extended Lessons in Data Warehousing
: Perform advanced data warehousing tasks using the Data Warehouse Center.

Development Center Tutorial for Video Online using Microsoft® Visual Basic
: Build various components of an application using the Development Center Add-in for Microsoft Visual Basic.

Information Catalog Center Tutorial
: Create and manage an information catalog to locate and use metadata using the Information Catalog Center.

Video Central for e-business Tutorial
: Develop and deploy an advanced DB2 Web Services application using WebSphere® products.

Visual Explain Tutorial
: Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 Information Center accessed from a browser

The DB2® Information Center gives you access to all of the information you need to take full advantage of DB2 Universal Database™ and DB2 Connect™ in your business. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, the Information Catalog Center, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser has the following features if you view it in Netscape Navigator 6.1 or higher or Microsoft Internet Explorer 5. Some features require you to enable support for Java or JavaScript:

Regularly updated documentation
: Keep your topics up-to-date by downloading updated HTML.

Search
: Search all of the topics installed on your workstation by clicking **Search** in the navigation toolbar.

Integrated navigation tree
: Locate any topic in the DB2 library from a single navigation tree. The navigation tree is organized by information type as follows:
  - Tasks provide step-by-step instructions on how to complete a goal.
  - Concepts provide an overview of a subject.
  - Reference topics provide detailed information about a subject, including statement and command syntax, message help, requirements.

Master index
: Access the information installed from the *DB2 HTML Documentation CD* from the master index. The index is organized in alphabetical order by index term.

Master glossary
: The master glossary defines terms used in the DB2 Information Center. The glossary is organized in alphabetical order by glossary term.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 290
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 292
- "Updating the HTML documentation installed on your machine" on page 294

Appendix D. 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/region 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/region 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

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Index

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

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) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Part Number: CT17VNA

Printed in U.S.A.

SC09-4822-00

(1P) P/N: CT17VNA

Spine information:

IBM® DB2 Universal Database™ Administration Guide: Planning

Version 8