IBM® DB2® Universal Database

Administration Guide: Performance

Version 7

SC09-2945-00

IBM® DB2® Universal Database

Administration Guide: Performance

Version 7

SC09-2945-00

Before using this information and the product it supports, be sure to read the general information under "Appendix G. Notices" on page 609.

Contents

About This Book

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

- Information about database design (found in *Administration Guide: Planning*)
- Information about implementing and managing databases (found in *Administration Guide: Implementation*)
- Information about configuring and tuning your database environment to improve performance (found in *Administration Guide: Performance*).

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

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

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

- The `Script Center` to store small applications called scripts. These scripts may contain SQL statements, DB2 commands, as well as operating system commands.

- The `Alert Center` to monitor the messages that result from other DB2 operations.
- The `Tool Settings` to change the settings for the Control Center, Alert Center, and Replication.
- The `Journal` to schedule jobs that are to run unattended.
- The `Data Warehouse Center` to manage warehouse objects.

Who Should Use This book

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

How This Book is Structured

This book contains information about the following major topics:

Introduction to Performance

- Chapter 1. Elements of Performance, introduces concepts and considerations for managing and improving DB2 UDB performance.
- Chapter 2. Architecture and Processes Overview, introduces underlying DB2 Universal Database architecture and processes.

Tuning Application Performance

- Chapter 3. Application Considerations, describes some techniques for improving database performance when designing your applications.
- Chapter 4. Environmental Considerations, describes some techniques for improving database performance when setting up your database environment.
- Chapter 5. System Catalog Statistics, describes how statistics about your data can be collected and used to ensure optimal performance.
- Chapter 6. Understanding the SQL Compiler, describes what happens to an SQL statement when it is compiled using the SQL compiler.
- Chapter 7. SQL Explain Facility, describes the Explain facility, which allows you to examine the choices the SQL compiler has made to access your data.

Tuning and Configuring Your System

- Chapter 8. Operational Performance, provides an overview of how the database manager uses memory and other considerations that affect run-time performance.

- Chapter 9. Using the Governor, provides an introduction to the use of a governor to control some aspects of database management.
- Chapter 10. Scaling Your Configuration Through Adding Processors, introduces some considerations and tasks associated with increasing the size of your database systems.
- Chapter 11. Redistributing Data Across Database Partitions, discusses the tasks required in a partitioned database environment to redistribute data across partitions.
- Chapter 12. Benchmark Testing, provides an overview of benchmark testing and how to perform benchmark testing.
- Chapter 13. Configuring DB2, discusses the database manager and database configuration files and the values for the configuration parameters.

Appendixes

- Appendix A. DB2 Registry and Environment Variables, presents profile registry values and environment variables.
- Appendix B. Explain Tables and Definitions, provides information about the tables used by the DB2 Explain facility and how to create those tables.
- Appendix C. SQL Explain Tools, provides information on using the DB2 explain tools: db2expln and dynexpln.
- Appendix D. db2exfmt - Explain Table Format Tool, formats the contents of the DB2 explain tables.
- Appendix F. Using the DB2 Library, provides information about the structure of the DB2 library, including wizards, online help, messages, and books.

A Brief Overview of the Other Volumes of the Administration Guide

Administration Guide: Planning

The *Administration Guide: Planning* is concerned with database design. It presents logical and physical design issues; distributed transaction issues; and high availability topics. The specific chapters and appendixes in that volume are briefly described here:

The World of DB2 Universal Database

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

Database Concepts

- ″Basic Relational Database Concepts″ presents an overview of database objects, including recovery objects, storage objects, and system objects.

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

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

Distributed Transaction Processing
- ″Designing Distributed Databases″ discusses how you can access multiple databases in a single transaction.
- ″Designing for Transaction Managers″ discusses how you can use your databases in a distributed transaction processing environment, such as CICS.

High Availability Systems
- ″Designing for High Availability″ presents an overview of the high availability failover support that is provided by DB2.
- ″High Availability Cluster Multi-Processing, Enhanced Scalability (HACMP ES) for AIX″ discusses DB2 support for high availability failover recovery on AIX.
- ″High Availability in the Windows NT Environment″ discusses DB2 support for high availability failover recovery on Windows NT.
- ″DB2 and High Availability on Sun Cluster 2.2″ discusses DB2 support for high availability failover recovery on the Sun Solaris operating system.

Appendixes
- ″Planning Database Migration″ provides information about migrating databases to Version 7.
- ″Incompatibilities Between Releases″ presents the incompatibilities introduced from release to release up to, and including, Version 7.
- ″National Language Support (NLS)″ introduces DB2 National Language Support, including information about countries, languages, and code pages.

Administration Guide: Implementation

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

Administering Using the Control Center

- ″Administering DB2 Using GUI Tools″ introduces the Graphical User Interface (GUI) tools used to administer the database.

Implementing Your Design

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

Database Security

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

Moving Data

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

Recovery

- ″Recovering a Database″ discusses factors to consider when choosing database and table space recovery methods, including backing up and restoring a database or table space, and using the roll-forward recovery method.

Appendixes

- ″Using Distributed Computing Environment (DCE) Directory Services″ provides information about how you can use DCE Directory Services.
- ″User Exit for Database Recovery″ discusses how user exit programs can be used with database log files, and describes some sample user exit programs.
- ″Issuing Commands to Multiple Database Partition Servers″ discusses the use of the *db2_all* and *rah* shell scripts to send commands to all partitions in a partitioned database environment.

- ″How DB2 for Windows NT Works with Windows NT Security″ describes how DB2 works with Windows NT security.
- ″Using the Windows NT Performance Monitor″ provides information about registering DB2 with the Windows NT Performance Monitor, and using the performance information.
- ″Working with Windows NT or Windows 2000 Database Partition Servers″ provides information about the utilities available to work with database partition servers on Windows NT or Windows 2000.
- ″Configuring Multiple Logical Nodes″ describes how to configure multiple logical nodes in a partitioned database environment.
- ″High Speed Inter-node Communications″ describes how to enable Virtual Interface Architecture for use with DB2 Universal Database.
- ″Lightweight Directory Access Protocol (LDAP) Directory Services″ provides information about how you can use LDAP Directory Services.
- ″Extending the Control Center″ provides information about how you can extend the Control Center by adding new tool bar buttons including new actions, adding new object definitions, and adding new action definitions.

Part 1. Introduction to Performance

Chapter 1. Elements of Performance

Performance is the way a computer system behaves given a particular work load. Performance is measured through one or more of the system's response time, throughput, and availability; and it is affected by:

- The resources available
- How well those resources are used and shared.

In general, you should undertake performance tuning when you want to improve the cost-benefit ratio of your system. Specific goals could include:

- Processing a larger, or more demanding, work load without increasing processing costs. (For example, increasing the work load without buying new hardware or using more processor time.)
- Obtaining faster system response times, or higher throughput, without increasing processing costs.
- Reducing processing costs without negatively affecting service to your users.

Translating performance from technical terms to economic terms is difficult. Performance tuning certainly costs money (through people's time and through processor time), so before you undertake a tuning project, weigh its costs against its possible benefits. Some of these benefits are tangible:

- More efficient use of resources
- The ability to add more users to the system.

Other benefits such as greater user satisfaction because of quicker response time, are intangible. All of these benefits should be considered.

There are wizards integrated with DB2 that will assist you in completing some performance-related administration tasks. These tasks are typically those where you spend a little time and can achieve a significant performance improvement. The wizards take you through each task one step at a time. Wizards are available through the Control Center and the Client Configuration Assistant.

The Performance Configuration wizard assists you to tune the performance of a database by updating configuration parameters to match your business requirements. This wizard, and, to a less extent the Create Database wizard, can assist in improving the performance of a database. Other wizards are available to assist in the improvement of performance of individual tables and general data access. The wizards in this area include: Create Table, Index, and

Configure Multisite Update wizards. The wizards can be found from the Control Center by clicking with the right mouse button on an object.

Tuning Guidelines

The following guidelines should help you develop an overall approach to performance tuning.

Remember the Law of Diminishing Returns: Your greatest performance benefits usually come from your initial efforts. Further changes generally produce smaller and smaller benefits and require more and more effort.

Do Not Tune Just for the Sake of Tuning: Tune to relieve identified constraints. If you tune resources that are not the primary cause of performance problems, this has little or no effect on response time until you have relieved the major constraints, and it can actually make subsequent tuning work more difficult. If there is any significant improvement potential, it lies in improving the performance of the resources that are major factors in the response time.

Consider the Whole System: You can never tune one parameter or system in isolation. Before you make any adjustments, consider how it will affect the system as a whole.

Change One Parameter at a Time: Do not change more than one performance tuning parameter at a time. Even if you are sure that all the changes will be beneficial, you will have no way of evaluating how much each change contributed. You also cannot effectively judge the trade-off you have made by changing more than one parameter at a time. Every time you adjust a parameter to improve one area, you almost always affect at least one other area that you may not have considered.

Measure and Reconfigure by Levels: For the same reasons that you should only change one parameter at a time, tune one level of your system at a time. You can use the following list of levels within a system as a guide:

- Hardware
- Operating System
- Application Server and Requester
- Database
- SQL Statements
- Application Programs

Check for Hardware and Software Problems: Some performance problems may be corrected by applying service either to your hardware, or to your software, or to both. Do not spend excessive time monitoring and tuning your system when simply applying service may make it unnecessary.

Understand the Problem Before You Upgrade Your Hardware: Even if it seems that additional storage or processor power could immediately improve performance, take the time to understand where your bottlenecks are. You may spend money on additional disk storage only to find that you do not have the processing power or the channels to exploit it.

Put Fallback Procedures in Place Before You Start Tuning: As noted earlier, some tuning can cause unexpected performance results. If this leads to poorer performance, it should be reversed and alternative tuning tried. If the former setup is saved in such a manner that it can be simply recalled, the backing out of the incorrect information becomes much simpler.

Disk Storage

We have already mentioned that the hardware that makes up your system can influence the performance of your system. As an example of the influence of hardware's on performance, we will consider some of the implications associated with disk storage.

How you manage disk storage affects performance in four ways:

- **How Storage is Divided**:

  How you divide a limited amount of storage between indexes and data, among table spaces, and among buffer pools, determines to a large degree how each will perform in different situations.
- **Wasted Storage**:

  Wasted storage in itself may not affect the performance of the system that is using it, but it may represent a resource that could be used to improve performance elsewhere.
- **Distributing Disk I/O**:

  How well you balance the demand for disk I/O across several disk storage devices, and controllers can affect how fast the database manager can retrieve information from disks.
- **Running Out of Storage**:

  Reaching the limit of available storage can degrade overall performance.

Performance Improvement Process

Use the following process to improve the performance of any system:

1. Establish performance indicators.
2. Define performance objectives.
3. Develop a performance monitoring plan.
4. Carry out the plan.

5. Analyze your measurements to determine whether you have met your objectives. If you have, consider reducing the number of measurements you make because performance monitoring itself uses system resources. Otherwise, continue with the next step.
6. Determine the major constraints in the system.
7. Decide where you can afford to make trade-offs and which resources can bear additional load. (Nearly all tuning involves trade-offs among system resources and the various elements of performance.)
8. Adjust the configuration of your system. If you think that it is feasible to change more than one tuning option, implement one at a time. If there are no options left at any level, you have reached the limits of your resources and need to upgrade your hardware.
9. Return to Step 4 above and continue to monitor your system.

Periodically, or after significant changes to your system or work load:

- Return to Step 1 above.
- Re-examine your objectives and indicators.
- Refine your monitoring and tuning strategy.

How Much Can a System be Tuned?

There are limits to how much you can improve the efficiency of a system. Consider how much time and money you should spend on improving system performance, and how much the spending of additional time and money will help the users of the system.

Your system may perform adequately without any tuning at all, but it probably will not perform to its potential. Each database is unique. As soon as you develop your own database, and applications to use it, investigate the tuning parameters available and learn how you can customize their settings to reflect your situation. In some circumstances, there will only be a small benefit from tuning a system; however, in most circumstances, the benefit may be significant.

Wizards are available from within the Control Center to assist in tuning the database parameters. The Performance Configuration wizard can be found by clicking the right mouse button on the database you want to tune from the Control Center.

As your system encounters a performance bottleneck, it is more likely that tuning will be effective. If you are close to the performance limits and you increase the number of users on the system by about ten percent, the response time is likely to rise by much more than ten percent. In this situation, you will need to determine how to counterbalance this degradation in performance by tuning your system. However, there is a point beyond which tuning cannot help you. At that point, you should consider revising your goals and expectations within that environment. Or, you should change your system environment by considering: more disk storage, faster CPU, additional CPUs, more main memory, faster communication links, or a combination of these changes.

A Less Formal Approach

If you do not have enough time to set performance objectives and to monitor and tune in a comprehensive manner, you can address performance by listening to your users. Find out if they are having performance-releated problems. You can usually locate the problem, or determine where to start looking for the problem, by asking a few simple questions. For example, you can ask your users:

- What do you mean by “slow response”? Is it ten percent slower than you expect it to be, or tens of times slower?
- When did you notice the problems? Is it recent or has it always been there?
- Do you know of other users who are complaining of the same problem? Are those complaining one or two individuals or a whole group?
- (If a whole group of users are experiencing difficulties, are they connected to the same terminal controller?)
- Are the problems you are experiencing related to a specific transaction or application program?
- Do your problems appear during regular periods such as at lunch hour, or are they continuous?

Putting It All Together

The underlying architecture of DB2 is important since an understanding of key concepts and processes will assist you with other performance issues. Topics such as storage architecture, data management, the processing model, and the memory model are all initially presented in the next chapter. See “Chapter 2. Architecture and Processes Overview” on page 11 for more information.

Tuning application performance is concerned with those performance topics associated with your applications and their interaction with the database. There are topics specific to applications themselves: Concurrency, Locking, Optimization Classes, control of results sets on queries, row blocking, use of compound SQL. In addition, there are brief discussions of: Character conversion as it relates to application performance; stored procedures; activation of databases; and the advantages of parallel processing. See “Chapter 3. Application Considerations” on page 41 for more information.

There are topics specific to optimization of queries: Configuration parameters affecting query optimization, the impact of node groups and table spaces on query optimization, and the large impact that indexes can have on query optimization. See “Chapter 4. Environmental Considerations” on page 87 for more information.

System catalog statistics have a significant influence on how well data is accessed by applications. The following topics are associated with statistics: The RUNSTATS utility, distribution statistics, index statistics, and those statistics that can be updated by users. See “Chapter 5. System Catalog Statistics” on page 107 for more information.

The SQL compiler takes each application and determines the best access plan for that application. Each query within the application is evaluated and may undergo several different operations designed to most clearly define the goal of the query. Then different methods of access (scans and joins) are reviewed for each query to determine the quickest way to retrieve the data requested by the query. The affects of parallelism are also considered. See “Chapter 6. Understanding the SQL Compiler” on page 139 for more information.

There are different tools available within the DB2 product to assist in the understanding of what is happening with the queries of an application. These tools are concerned with explaining what is affecting application performance. See “Chapter 7. SQL Explain Facility” on page 201 for more information.

In addition to tuning individual applications, you should also consider the performance of the database where those applications are running. Performance of your database is determined in large part by how well memory is used. There are many topics surrounding memory that are concerned with performance: buffer pools, prefetching of data, parallel I/O, sorting capabilities, the need to reorganize the data in tables, and the concept of database agents. See “Chapter 8. Operational Performance” on page 227 for more information.

There is a Governor that can be set up to manage how applications are using the database. See “Chapter 9. Using the Governor” on page 267 for more information.

The number of processors and the number of database partitions can be increased to improve the performance of the database. See “Chapter 10. Scaling Your Configuration Through Adding Processors” on page 281 for more information.

Once you have increased the number of database partitions, you will want to ensure the data in the database is spread or redistributed correctly among the database partitions. See “Chapter 11. Redistributing Data Across Database Partitions” on page 291 for more information.

To determine how well your database is performing, you can conduct benchmark testing. The methodology for benchmark testing, how to prepare for a benchmark test, the creation of a benchmark program, and the running of benchmark tests are all topics of importance. See “Chapter 12. Benchmark Testing” on page 299 for more information.

The very extensive set of database manager and database configuration parameters are presented individually within “Chapter 13. Configuring DB2” on page 311.

There is additional information that is related to these performance topics. The appendices include the following:
- “Appendix A. DB2 Registry and Environment Variables” on page 469
- “Appendix B. Explain Tables and Definitions” on page 497
- “Appendix C. SQL Explain Tools” on page 531
- “Appendix D. db2exfmt - Explain Table Format Tool” on page 577
- “Appendix E. Configuring XA Transaction Managers to Use DB2 UDB” on page 579
- “Appendix F. Using the DB2 Library” on page 591

Chapter 2. Architecture and Processes Overview

When working with the performance of the database operations for DB2, you need some understanding of the rudimentary concepts involving the DB2 architecture and processes. This chapter presents sufficient information to provide you with information on how DB2 Universal Database works. While later chapters provide greater detail on some of the topics found here, what is shown in this chapter creates the context for later understanding.

The first figure shows an overview of the architecture and processes for DB2 UDB.

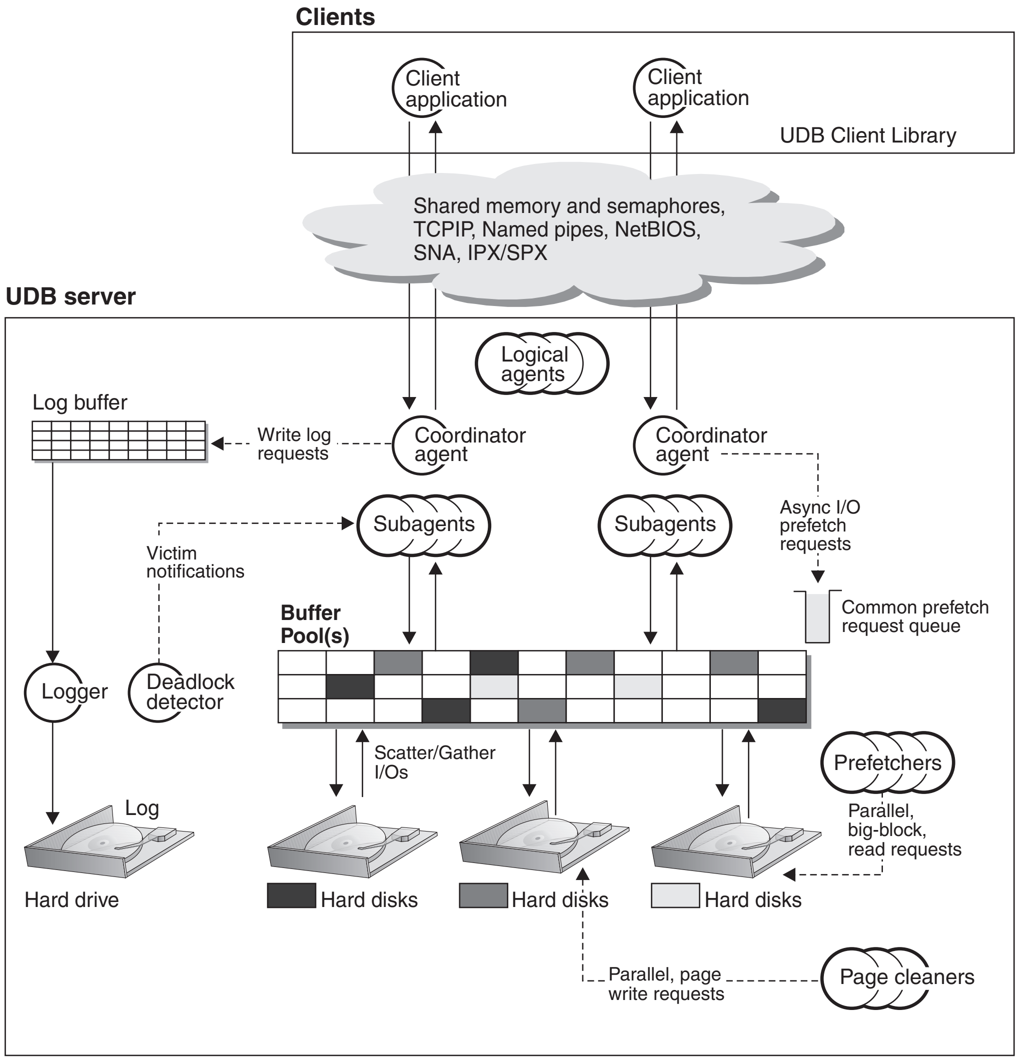


Figure 1. Architecture and Processes Overview

On the client-side, there are local and/or remote applications that are linked with the DB2 Universal Database client library.

Between the clients and the DB2 Universal Database server is a “cloud” representing the means of communication between the local or remote clients, and the server. Local clients communicate using shared memory and semaphores; remote clients use a protocol such as Named Pipes (NPIPE), TCP/IP, NetBIOS, IPX/SPX, or SNA.

On the server-side, activity is controlled by **engine dispatchable units** (EDUs). In all figures in this chapter, EDUs are shown as circles or groups of circles. EDUs are implemented as threads on Windows NT and on OS/2 (all within a single process), and as processes on UNIX. The most common type of EDUs are DB2 agents. These EDUs carry out the bulk of the SQL processing on behalf of applications. Other examples of EDUs are the DB2 prefetchers and page cleaners which are responsible for various types of I/O processing. See "Database Agents" on page 257 for more information.

Each client application is assigned a unique EDU called a "coordinator agent" which coordinates the processing for that application and communicates with it. There may also be a set of subagents assigned together to work on processing the client application requests. Multiple subagents maybe assigned so that if the machine where the server resides has multiple processors, like in a symmetric multiprocessing environment, the client application requests can exploit those processors.

All agents and subagents are managed using a pooling algorithm which minimizes the creation and/or destruction of EDUs.

A buffer pool is an area of storage memory where database pages of user table data, index data, and catalog data are temporarily moved and perhaps modified. The buffer pool is a key influencer of overall database performance because data can be accessed much faster from memory than from a disk. If more of the data needed by applications were present in the buffer pool then less time would be needed to access this data compared to time taken to find the data out on disk storage. See "Managing the Database Buffer Pool" on page 235 for more information.

The configuration of the buffer pool, along with prefetcher and page cleaner EDUs, control the availability of the data needed by the applications.

The prefetchers are present to retrieve data from disk and move it into the buffer pool before applications need the data. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk into the buffer pool if there were no data prefetchers. Agents of the application send asynchronous read-ahead requests to a common prefetch queue. As prefetchers become available, they implement those requests by using big-block or scatter read input operations to bring the requested pages from disk to the buffer pool. Having multiple disks for storage of the database data means that the data can be striped across the disks. This striping of data enables the prefetchers to use multiple disks at the same time to retrieve data. See "Prefetching Data into the Buffer Pool" on page 241 and "Configuring I/O Servers for Prefetching and Parallel I/O" on page 244 for more information.

Prefetchers are used to bring data into the buffer pool. Page cleaners are used to move data from the buffer pool back out to disk.

Page cleaners are background EDUs, independent of the application agents, that look for, and write out, pages from the buffer pool that are no longer needed. Page cleaners can ensure that there is room in the buffer pool for the pages being retrieved by the prefetchers.

Without the existence of the independent prefetchers and page cleaner EDUs, the application agents would have to do all of the reading and writing of data between the buffer pool and disk storage.

With multiple applications working with data from the database there are opportunities for a "deadlock" to occur between two or more applications. A deadlock is illustrated in the following figure.

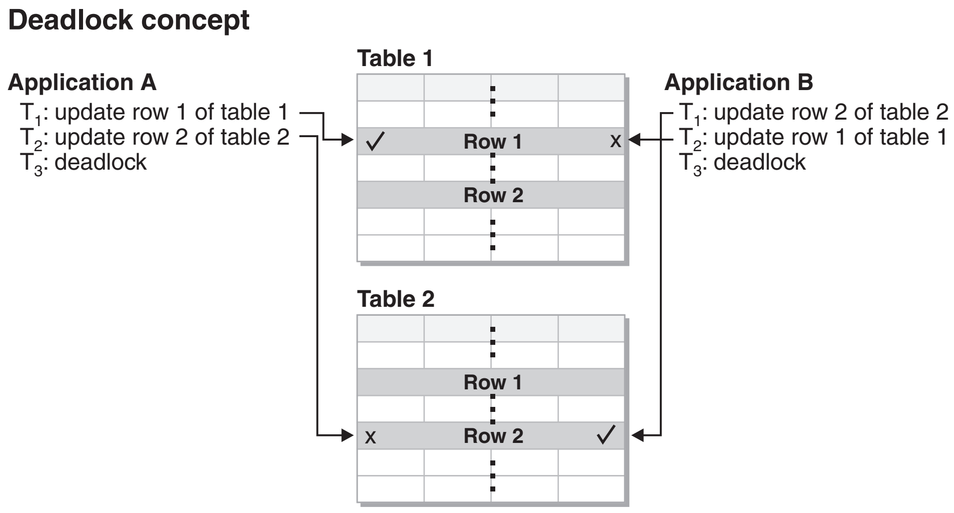

Deadlock concept
Application A
T1: update row 1 of table 1
T2: update row 2 of table 2
T3: deadlock
Table 1
Row 1
Row 2
Table 2
Row 1
Row 2
Application B
T1: update row 2 of table 2
T2: update row 1 of table 1
T3: deadlock

Figure 2. deadlock Detector

A "deadlock" means that more than one application is waiting for another application to release a lock on data. Each of the waiting applications is holding data needed by other applications through locking. This locked data is required by one or more other applications which are, in turn, holding data needed by other applications. Mutual waiting for the other application to release a lock on held data leads to a deadlock: The applications can wait forever until the "other" application releases the lock on the held data. The other applications do not voluntarily release locks on data that they need. A process is required to break these deadlock situations.

As its name suggests, the deadlock detector monitors the information about agents waiting on locks. The deadlock detector arbitrarily selects one of the applications in the deadlock to release the locks currently held by the "volunteered" application. By releasing the locks of that application, the data required by other waiting applications is made available for use. The formerly waiting applications are then free to use the data required to complete actions on data in the database.

Changes to data pages in the buffer pool are logged. A log buffer exists and is associated with a logger EDU. Agents updating a data record in the database update the associated page in the buffer pool and write a log record. The log record contains the information necessary to either redo or undo the change. Neither the page in the buffer pool nor the log record in the log buffer are written to disk immediately to optimize performance. The logger EDU and the buffer pool manager cooperate to implement a Write Ahead Logging (WAL) protocol that ensures that the data page is not written to disk before its associated log record is written to the log. The WAL protocol ensures that there is always enough information in the log to recover from a crash and to restore database consistency. If an uncommitted update on a page was written to a disk, crash recovery uses the undo information in the associated log record to undo the update. If a committed update did not make it to disk, crash recovery uses the redo information in the associated log record to redo the update.

Note: On a COMMIT, all log records in the transaction are flushed to disk, if they were not already flushed.

Storage Architecture

Within the discussion of storage architecture, we will consider:

- "Database Directory"
- "Table Spaces" on page 17

Database Directory

When you create a database, information about the database including default information is placed within a directory. The directory structure is created for you at a location that is based on the information you provide in the CREATE DATABASE command. If you do not specify the location of the path or drive when creating the database, the default location is used.

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

At the directory you specify in the CREATE DATABASE command, a subdirectory using 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. Following the instance name subdirectory, a subdirectory using “NODE0000” is created. This subdirectory is used to differentiate partitions in a multiple logical partitioned database environment. Following the node directory, a subdirectory using “SQL00001” is created. This subdirectory is named using the database token and represents the database being created. It is also used to differentiate databases created in this instance on the directory you specified in the CREATE DATABASE command.

The directory structure would appear like the following:

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

The database directory will contain several files that were created as part of the CREATE DATABASE command. Buffer pool information is contained in the files: SQLBP.1 and SQLBP.2. Table space information is contained in the files: SQLSPCS.1 and SQLSPCS.2. There are two of each of these files to allow for backing up the information in these files.

Database configuration information is contained in: SQLDBCON. The history file *db2rhist.asc* and its backup *db2rhist.bak* are readable by you and contain history information about backups, restores, loading of tables, reorganization of tables, altering of a table space, and other changes to a database.

The log control file, SQLOGCTL.LFH, contains 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. As well, this can provide a substantial peformance benefit, as the log files and database containers are not competing for movement of the same disk heads. You can change the location of the log subdirectory using the *newlogpath* database configuration parameter.

The SQLT* subdirectories are created and contain the default System Managed Space (SMS) table spaces required for an operational database. There are three default table spaces 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.

You will also read of “containers” when considering table spaces. For SMS table spaces, containers are operating system directories.

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 will not attempt to use these subdirectories. There are also other files that are created under the container subdirectories with different name extensions to distinguish between the type of data stored in the files. The extensions are:

- SQL*.DAT (containing non-long table data)
- SQL*.LF (containing LONG VARCHAR or LONG VARGRAPHIC data)
- SQL*.LB (containing BLOB, CLOB, or DBCLOB data)
- SQL*.LBA (containing allocation and free space information about SQL*.LB files)
- SQL*.INX (containing index table data)
- SQL*.DTR (containing temporary data for a reorganization of a SQL*.DAT file)
- SQL*.LFR (containing temporary data for a reorganization of a SQL*.LF file)
- SQL*.RLB (containing temporary data for a reorganization of a SQL*.LB file)
- SQL*.RBA (containing temporary data for a reorganization of a SQL*.LBA file)

Table Spaces

There are two types of table spaces supported: System Managed Space (SMS) and Database Managed Space (DMS). Each has its own characteristics that make it appropriate for different environments. Refer to *Administration Guide: Planning* for more information on designing and choosing table spaces.

SMS Table Spaces

System Managed Space (SMS) table spaces store data in operating system files. The data in the table spaces is striped by extent across all the containers in the system. An **extent** is a group of consecutive pages defined to the database. Each table in a table space is given its own file name which is used in all containers. The file extension denotes the type of the data stored in the file. The starting extent for each table is placed in “round robin” fashion throughout the containers. This spreads the space requirement evenly across all containers in the table space. This is very important when there are a large number of small tables.

Allocation of space is done when there is a demand for additional space. By default, space is allocated one page at a time.

DMS Table Spaces

With Database Managed Space (DMS) table spaces, the database manager controls the storage space. A list of devices or files is selected to belong to a table space when the DMS table space is defined. The space on those devices or files is managed by the DB2 database manager. As with SMS table spaces and containers, DMS table spaces and the database manager use striping by extent to ensure an even distribution of data across all containers.

DMS table spaces differ from SMS table spaces in that for DMS table spaces, space is allocated when the table space is created and not allocated when needed.

Also, placement of data can differ on the two types of table spaces. For example, consider the need for efficient table scans: It is important that the pages in an extent are physically contiguous. With SMS, the file system of the operating system decides where each logical file page is physically placed. The pages may, or may not, be allocated contiguously depending on the level of other activity on the file system and the algorithm used to determine placement. With DMS, however, the database manager can ensure the pages are physically contiguous because it interfaces with the disk directly.

There is one exception to this general statement regarding contiguous placement of pages in storage. There are two container options when working with DMS table spaces: Raw devices and files. When working with file containers, the database manager allocates the entire container at table space creation time. A result of this initial allocation of the entire table space is that the physical allocation is typically, but not guaranteed to be, contiguous even though the file system is doing the allocation. When working with raw device containers, the database manager takes control of the entire device and always ensures the pages in an extent are contiguous.

Unlike SMS table spaces, the containers that make up a DMS table space do not need to be close to being equal in their capacity. However, it is recommended that the containers are equal, or close to being equal, in their capacity. Also, if any container is full, any available free space from other containers can be used in a DMS table space.

When working with DMS table spaces, you should consider associating each container with a different disk. This allows for a larger table space capacity and the ability to take advantage of parallel I/O operations.

The next figure shows the logical address map for a DMS table space.

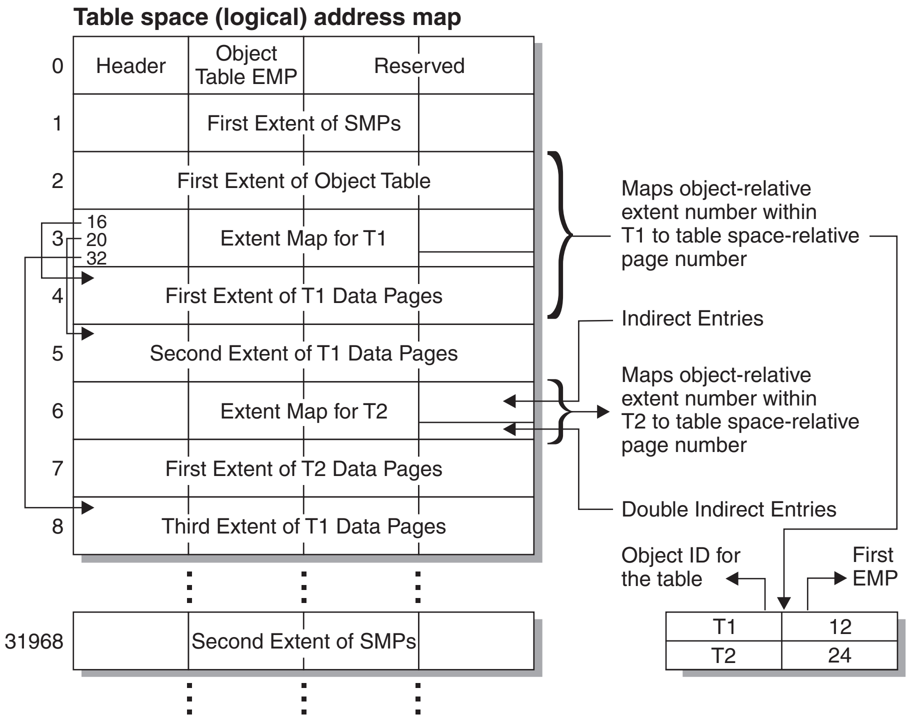


Table space (logical) address map
0
Header
Object Table EMP
Reserved
1
First Extent of SMPs
2
First Extent of Object Table
3
16
20
32
Extent Map for T1
4
First Extent of T1 Data Pages
5
Second Extent of T1 Data Pages
6
Extent Map for T2
7
First Extent of T2 Data Pages
8
Third Extent of T1 Data Pages
31968
Second Extent of SMPs
Maps object-relative extent number within T1 to table space-relative page number
Indirect Entries
Maps object-relative extent number within T2 to table space-relative page number
Double Indirect Entries
Object ID for the table
First EMP
T1
12
T2
24

Figure 3. DMS Table Spaces

The CREATE TABLESPACE statement creates a new table space within a database, assigns containers to the table space, and records the table space definition and attributes in the catalog. One of the things defined when creating the table space is the extent size. An extent is the unit of space allocation within a table space. It is simply a set of contiguous pages. The extent size is the number of contiguous pages. Only one table (or other object, such as an index) can use the pages in any single extent. All objects (tables, indexes, and others) created in the table space are allocated extents in a logical table space address map. An extent belongs to only one object at a time. Extent allocation is managed through Space Map Pages (SMP).

The first extent in the logical table space address map is a header for the table space containing internal control information. The second extent is the first extent of Space Map Pages (SMP) for the table space. SMP extents are spread at regular intervals throughout the table space. Each SMP extent is simply a bit map of the extents from the current SMP extent to the next SMP extent. The bit map is used to track which of the intermediate extents are in use or not.

The next extent following the SMP is the object table for the table space. The object table is an internal table that tracks which user objects exist in the table space and where their first Extent Map Page (EMP) extent is located. Each object has its own EMPs which provide a map to each page of the object that is stored in the logical table space address map.

The object table is an internal relational table that maps an object identifier to the location of the table's first EMP extent. This EMP extent, directly or indirectly, maps out all extents in the object. Each EMP contains an array of entries. Each entry maps an object-relative extent number to a table space-relative page number where the object extent is located. Direct EMP entries directly map object-relative addresses to table space-relative addresses. The last EMP page in the first EMP extent contains indirect entries. Indirect EMP entries map to EMP pages which then map to object pages. The last 16 entries in the last EMP page in the first EMP extent contain double-indirect entries.

The extents from the logical table space address map are striped in a round robin fashion across the containers associated with the table space.

Comparing SMS and DMS Table Spaces

When comparing SMS and DMS table spaces, SMS table spaces are an excellent choice for general purposes. SMS table spaces provide very good performance with very little administration cost. DMS table spaces are the best choice when seeking top performance. Device containers provide the best performance since double buffering can occur when moving data using file containers or SMS table spaces. (Double buffering can occur when the data is buffered first at the database manager level and then again at the file system level.)

Data Management

Following the creation of a database, the creation of a table space, the creation of a table, and the placing of data into the table, it is interesting to know how the table is organized and how indexes are used to retrieve that table data.

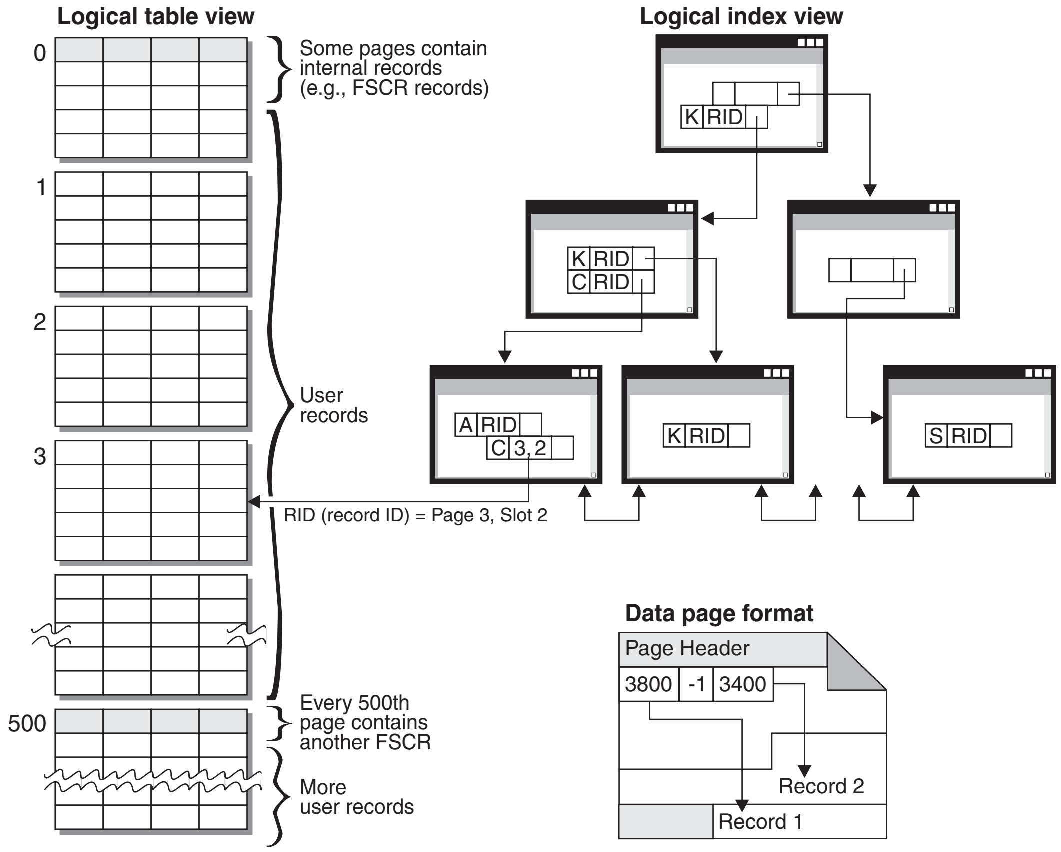


Figure 4. Tables, Records, and Indexes

Logically, table data is organized as a list of data pages. And these data pages are logically grouped together based on the extent size of the table space. For example, if the extent size is four, pages zero to three are part of the first extent, pages four to seven are part of the second extent, and so on.

The number of records contained within each data page can vary based on the size of the data page and the size of the records. A maximum of 255 records can fit on one page. Most pages contain only user records. However, a small number of pages include special internal records, that are used by DB2 to manage the table. For example, there is a Free Space Control Record (FSCR) on every 500th data page. These records map out how much free space for new records exists on each of the following 500 data pages (until the next FSCR). This available free space is used when inserting records into the table.

Logically, index pages are organized as a B-tree which can efficiently locate records in the table data which have a given key value. The number of entities on an index page is not fixed but depends on the size of the key. For tables in DMS table spaces, record identifiers (RIDs) in the index pages use table space-relative page numbers, not object-relative page numbers. This allows an **index scan** to directly access the data pages without requiring an Extent Map page (EMP) for mapping.

Each data page has the following format: A page header begins each data page. After the page header there is a slot directory. Each entry in the slot directory corresponds to a different record on the page. The entry itself is the byte-offset into the data page where the record begins. Entries of minus one (-1) correspond to deleted records.

Record Identifiers and Pages

Record identifiers (RIDs) are a three-byte page number followed by a one-byte slot number. Once the index is used to identify a RID, the RID is used to get to the correct data page and slot number on that page. The contents of the slot is the byte-offset within the page to the beginning of the record being sought. Once a record is assigned a RID, it does not change until a table reorganization.

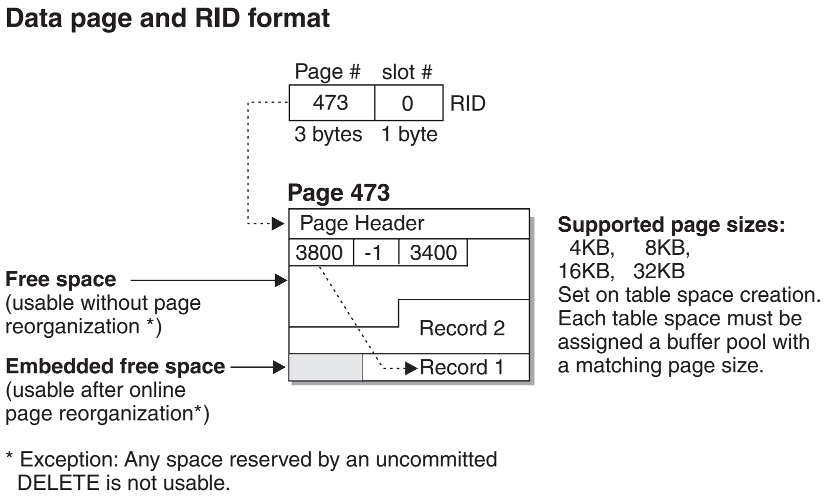

Data page and RID format
Page #
slot #
473
0
RID
3 bytes
1 byte
Page 473
Page Header
3800
-1
3400
Record 2
Record 1
Free space
(usable without page reorganization *)
Embedded free space
(usable after online page reorganization*)
Supported page sizes:
4KB, 8KB, 16KB, 32KB
Set on table space creation. Each table space must be assigned a buffer pool with a matching page size.
* Exception: Any space reserved by an uncommitted DELETE is not usable.

Figure 5. Data Page and RID Format

When a table is reorganized, embedded free space that is left on the data page following the deletion of a record is converted to usable free space. RIDs are redefined based on movement of records on a data page to take advantage of the usable free space.

DB2 supports different page sizes. Use larger page sizes for workloads that tend to access rows sequentially. For example, sequential access is used for Decision Support applications or where temporary tables are extensively used.

Use smaller page sizes for workloads that tend to be more random in their access. For example, random access is used in OLTP-environments.

For more information on reorganizing a table, see “Reorganizing Catalogs and User Tables” on page 251.

Space Management

You use the SQL INSERT statement to place new information into a table. When you do this, there is an INSERT search algorithm that is followed to complete the work. First the Free Space Control Records (FSCRs) are used to find a page with enough space. However, even when the FSCR says a page has enough free space, the space may not be usable because it is “reserved” by an uncommitted DELETE from another transaction. As a result, you should ensure that all transactions COMMIT frequently otherwise uncommitted freed space will not be usable.

Not all FSCRs in a table are searched. The DB2MAXFSCRSEARCH registry variable limits the number of FSCRs considered when attempting an INSERT. The default value for this registry variable is five. If no space is found within five FSCRs, then the record being inserting is appended to the end of the table. And, to optimize INSERT speed, subsequent records are also appended to the end of the table until two extents are filled. Once the two extents are filled, the next INSERT resumes searching at the FSCR where the last search ended.

Note: The value of DB2MAXFSCRSEARCH is important. To optimize for INSERT speed (at the possible expense of quicker table growth), set this registry variable to a small value. To optimize for space reuse (at the possible expense of INSERT speed), set this registry variable to a large value.

Once the entire table is searched, the record to be inserted will be appended without additional searching. Searching using the FSCRs is not done again until space is created some where in the table (following a DELETE, for example).

There are two other search algorithm options. The first is APPEND MODE. In this mode, new rows are always appended to the end of the table. No searching or maintenance of FSCRs takes places. This option is enabled using the ALTER TABLE APPEND ON statement, and can increase performance for tables that only grow, like journals. The second choice is to define a clustering index on the table. In this case, the database manager attempts to insert records on the same page as other records with similar index key values. If there is no space on that page, the attempt is made to put the record into the surrounding pages. If there is still no success, the FSCR search algorithm, described above, is used – with one small difference: a worst-fit approach is used rather than a first-fit approach. This worst-fit approach tends to choose pages with more free space. This is done to establish a new clustering area for rows with this key value.

When you define a clustering index on a table, use ALTER TABLE... PCTFREE before either loading or reorganizing the table. The PCTFREE clause leaves the percentage value given as free space on that table's data page after loading and reorganizing. This increases the likelihood that the cluster index operation will find free space on the desired page.

Data pages and overflow records

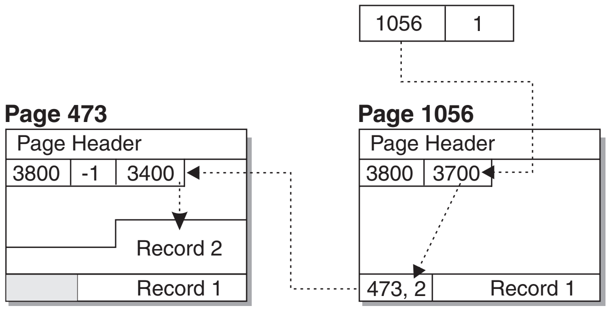


Figure 6. Data Page and Overflow Records

Overflow records are possible when an update request enlarges an existing record so that it cannot fit into the current page. The enlarged record is inserted on another page, where there is sufficient room, as an overflow record. The original RID is converted to a pointer record which contains the new RID of the overflow record. The indexes for the table keep the original RID and an extra page read is required to get to the data record requested. Many overflow records means many extra page reads and slower performance accessing the table. Reorganization of the table eliminates overflow records. Whenever possible, however, you should avoid update requests that enlarge records and so avoid overflow records.

Index Management

DB2 indexes are an optimized B-tree implementation based on an efficient and high concurrency index management method using write-ahead logging.

The optimized B-tree implementation has bi-directional pointers on the leaf pages that allows a single index to support scans in either forward or reverse direction (but not both at the same time). Index page splits are normally right in half except at the high-key page where a 90/10 split is used. That is, the high ten percent of the index keys are placed on a new page. This type of index page split is useful for workloads where INSERT requests are often completed with new high-keys.

Pages in the index are freed when the last index key on the page is removed. The exception to this rule occurs when the MINPCTUSED clause is selected when creating the index. The use of this clause indicates that the index can be reorganized online; and that the value given with this clause is the threshold for the minimum percentage of space used on the index leaf pages. If, after a key is deleted from an index page, the percentage of space used on the page is at or below the value given then an attempt is made to merge the remaining keys with those of a neighboring page. If there is sufficient room, the merge is performed and an index leaf page is deleted. Use of this clause can improve space reuse; however, if the value used is too high then the time taken to attempt a merge increases but also becomes less likely to succeed. It is recommended that the value for this clause always be less than fifty percent.

The INCLUDE clause of the CREATE INDEX statement allows for the inclusion of the specified column(s) on the index leaf pages in addition to the key columns. This can increase the number of queries that are eligible for index-only access. However, this can also increase the index space requirements and, possibly, index maintenance costs if the included columns are updated frequently. Ordering the index B-tree is only done using the key columns and not the included columns.

Locking

The database manager provides concurrency control and prevents uncontrolled access to resources and data by means of locks. A **lock** associates an application with a database manager resource or data record. The lock controls how other applications can access the same resource or data record.

The database manager uses record-level locking or table-level locking as appropriate based on:

- The isolation level specified at precompile time or when an application is bound to the database. The isolation level can be one of: Uncommitted Read (UR), Cursor Stability (CS), Read Stability (RS), or Repeatable Read (RR). The different isolation levels are used to control access to uncommitted data, prevention of lost updates, allowance of non-repeatable reads of data, and prevention of phantom reads. Use the minimum isolation level that satisfies your application needs.
- The access plan selected by the optimizer. Table scans, index scans, and other methods of data access each require different types of access to the data.
- The table’s LOCKSIZE attribute. The LOCKSIZE clause on the ALTER TABLE statement indicates the granularity of the locks used when the table is accessed. The choices are either ROW for row locks, or TABLE for table locks. Use ALTER TABLE... LOCKSIZE TABLE for read-only tables. This reduces the number of locks required by database activity.

- The amount of memory devoted to locking. The amount of memory devoted to locking is controlled by the *locklist* database configuration parameter. If the lock list fills, performance can degrade due to lock escalations and reduced concurrency on shared objects in the database. Increase the value of *locklist* and/or *maxlocks* if you encounter frequent lock escalations.

Ensure all transactions COMMIT frequently to free held locks.

In general, record-level locking is used unless one of the following is the case:
- The isolation level chosen is Uncommitted Read (UR).
- The isolation level chosen is Repeatable Read (RR) and the access plan requires a scan with no predicates.
- The table's LOCKSIZE attribute is "TABLE".
- The lock list fills and lock escalation occurs.
- There is an explicit table lock acquired via the LOCK TABLE statement. The LOCK TABLE statement prevents concurrent application processes from either changing a table or using a table.

A **lock escalation** is the conversion of one or more record locks to a table lock. An exclusive lock escalation is a lock escalation where the table lock acquired is an exclusive lock. Lock escalations reduce concurrency and should be avoided.

The duration of record locking varies with the isolation level being used:
- Uncommitted Read (UR) scans: No record locks are held unless a record is changing.
- Cursor Stability (CS) scans: Record locks are only held while the cursor is positioned on the record.
- Read Stability (RS) scans: Only qualifying record locks are held for the duration of the transaction.
- Repeatable Read (RR) scans: All record locks are held for the duration of the transaction. If you are in an environment where this isolation level is not needed or not wanted, use the DB2_RR_TO_RS registry variable. This tells the database manager to avoid the extra locking required to enable RR semantics and, as a result, increase peformance.

See "Locking" on page 48 for more information on this topic.

Logging

There are two logging strategy choices:
- Circular logging where the log records fill the log files and then overwrite the initial log records in the initial log file. The overwritten log records are not recoverable.

- Retain log records where once a log file is filled with log records, it is archived. New log files are made available for log records. Retaining log files enables **roll-forward recovery**. Roll-forward recovery reapplies changes to the database based on completed units of work (transactions) that are recorded in the log. You can specify that roll-forward recovery is to the end of the logs, or to a particular point in time before the end of the logs.

No matter which choice is made, all changes to regular data and index pages are written to the log buffer. The data in the log buffer is only forced to disk:
- Before the corresponding data pages are being forced to disk. This is called “write-ahead logging”.
- On a COMMIT; or after the value of the number of COMMITS to group (*mincommit*) database configuration parameter is reached.
- When the log buffer is full. Double buffering is used to prevent I/O waits.

Note: At the time the transaction completes by using the COMMIT statement, all changed pages are flushed to disk to ensure recoverability.

When transactions are short, the log I/O can become a “bottleneck” due to the frequency of the flushing of the log at COMMIT time. In such environments, setting the *mincommit* configuration parameter to a value greater than one can remove the “bottleneck”. When a value greater than one is used, the COMMITs for several transactions are held or “batched”. The first transaction to COMMIT waits until (*mincommit* - 1) more transactions COMMIT; and then the log is forced to disk and all transactions respond to their applications. The result is only one log I/O instead of several individual log I/O’s.

In order to avoid an excessive degradation in response time, each transaction only waits up to one second for the (*mincommit* - 1) other transactions to COMMIT. If the one second of time expires, the waiting transaction will force the log itself and respond to its application. This allows you to set *mincommit* and yet not be too concerned with performance during times of fewer transactions being processed.

Changes to Large Objects (LOBs) and LONG VARCHARs are tracked through shadow paging. LOB column changes are not logged unless log retain is used and the LOB column is defined on the CREATE TABLE statement as not using the NOT LOGGED clause. Changes to allocation pages for LONG or LOB data types are logged like regular data pages.

What Happens When Updating

What happens to the log and to the data page when an agent updates a page? The protocol described here minimizes the I/O required by the transaction and also ensures recoverability.

First, the page to be updated is pinned and latched with an exclusive lock. A log record is written to the log buffer describing how to redo and undo the change. As part of this action, a log sequence number (LSN) is obtained and is stored in the page header of the page being updated. The change is then made to the page. Finally, the page is unlatched and unfixed. The page is considered to be “dirty” because there are changes to the page that have not been written out to disk. The log buffer has also been updated.

Both the data in the log buffer and the “dirty” data page will need to be forced to disk. For the sake of performance, these I/Os are delayed until a convenient point (for example, during a lull in the system load), or until necessary to ensure recoverability, or to bound recovery time. More specifically, a “dirty” page is forced to disk:

- When another agent chooses it as a victim.
- When a page cleaner acts on the page as the result of:
  - Another agent choosing it as a victim.
  - The *chngpgs_thresh* database configuration parameter percentage value is exceeded. Once exceeded, asynchronous page cleaners “wake-up” and write changed pages to disk.
  - The *softmax* database configuration parameter percentage value is exceeded. Once exceeded, asynchronous page cleaners “wake-up” and write changed pages to disk.
- When the page was updated as part of a table which has the NOT LOGGED INITIALLY clause invoked and a COMMIT is issued. At the time of the COMMIT all changed pages are flushed to disk to ensure recoverability.

A log buffer is forced to disk by the logger engine dispatchable unit (EDU):

- Before the corresponding data pages are being forced to disk. This is called “write-ahead logging”.
- On a COMMIT; or after the value of the number of COMMITS to group (*mincommit*) database configuration parameter is reached.
- When the log buffer is full. Double buffering is used to prevent I/O waits.

Process Model

Local and remote application processes can work with the same database. A remote application is one that initiates a database action from a machine that is remote from the database machine. Local applications are directly attached to the database at the server machine.

Each of the circles of the following figure represent engine dispatchable units (EDUs) which are known as “processes” on UNIX platforms, and “threads” on Windows NT and OS/2 platforms.

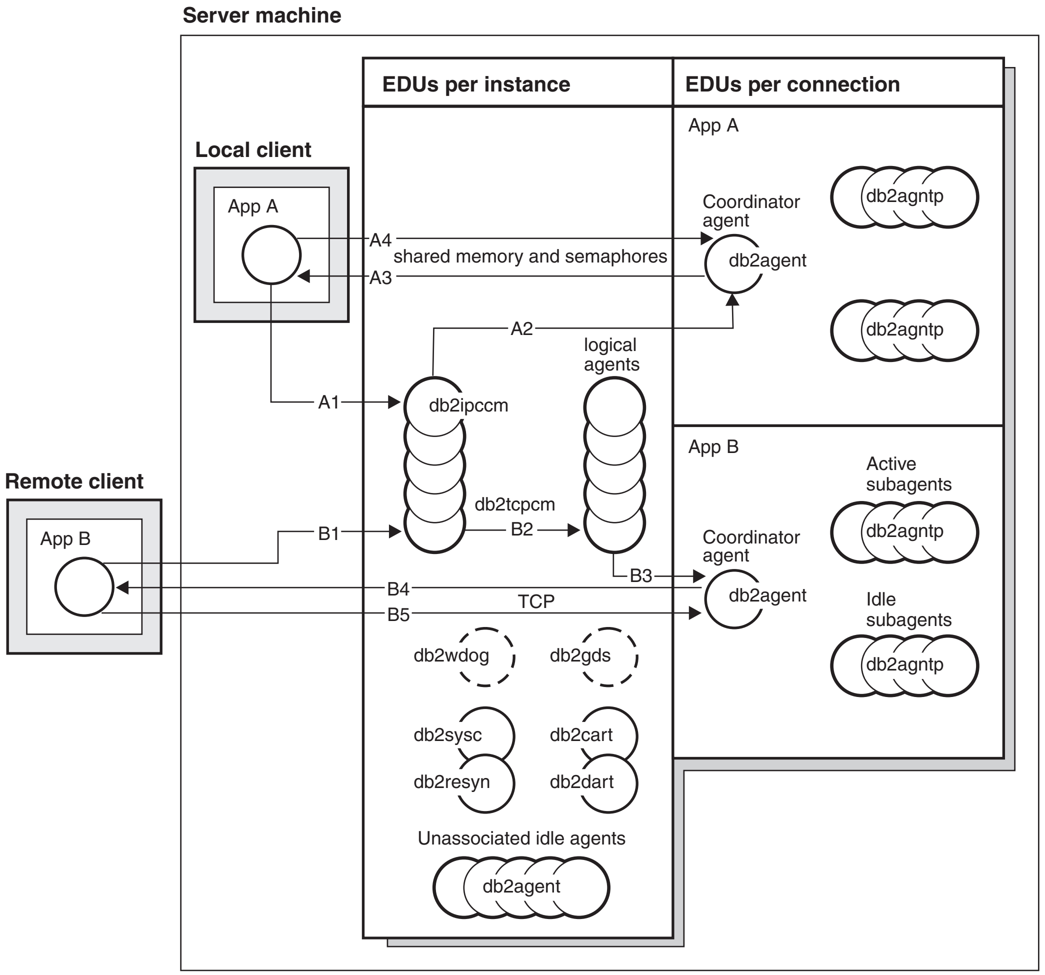


Figure 7. Process Model Overview

A means of communicating between an application and the database manager must be established before the work the application wants done at the database can be carried out.

In the figure above at A1, a local client establishes communications first by working with the db2ipccm engine dispatchable unit (EDU). This EDU at A2 works with a db2agent EDU that becomes the coordinator agent for the application requests from the local client. The coordinator agent contacts the client application at A3 and establishes shared memory and semaphores communication between the client application and the database at A4. The application at the local client is connected to the database.

In the figure above at B1, a remote client establishes communications first by working with the db2tcpcm EDU. If another communications protocol was chosen, the appropriate EDU would be used. The db2tcpcm EDU, at B2, works with a logical agent. This EDU at B3, works with a db2agent EDU that becomes the coordinator agent for the application requests from the remote client. The coordinator agent contacts the client application at B4 and establishes TCP/IP communication between the client application and the database at B5. The application at the remote client is connected to the database.

Other things to notice in this figure:

- There are two classes of agent: A logical agent and a worker agent. A logical agent represents a connected application to the database manager. A worker agent carries out application requests but has no permanent attachement to any particular application.
- There are four types of worker agents: Active coordinator agents, subagents, inactive agents, and idle agents.
- Each process or thread of a client application represented by a logical agent will be linked to an active coordinator agent.
- In a partitioned database environment, and enabled intra-partitioned parallelism environments, the coordinator agents distribute database requests to subagents (db2agntp). The subagents perform the requests for the application.
- There is an agent pool (db2agent) where idle agents wait for new work.
- There are other EDUs that manage client connections, logs, two-phase COMMITs, backup and restore tasks, and other tasks.

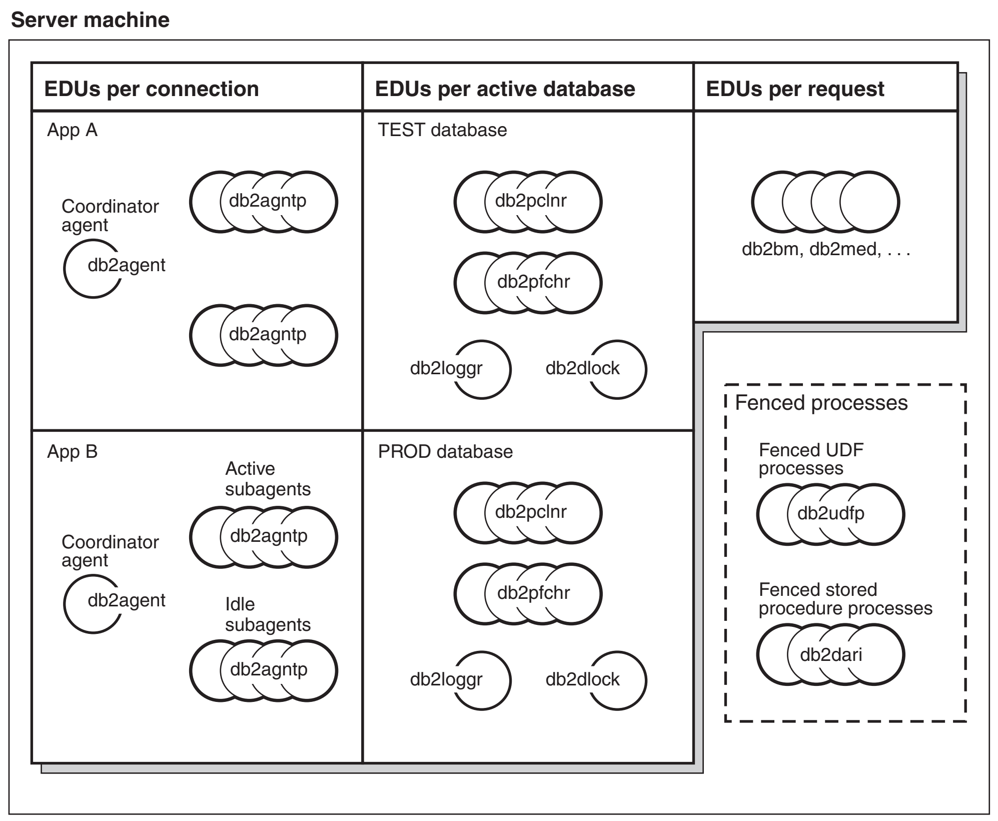


Figure 8. Process Model Part 2

This figure shows additional engine dispatchable units (EDUs) that are part of the server machine environment. Each active database has its own shared pool of prefetchers (db2pfchr) and page cleaners (db2pclnr), and its own logger (db2loggr) and deadlock detector (db2dlock).

The circles labelled “db2udfp” and “db2dari” at the botton right of the figure represent processes run within DB2 Universal Database as fenced User-defined Functions (UDFs) and Stored Procedures respectively. These processes are managed in order to minimize costs associated with their creation and destruction. The default for the *keepdari* database manager configuration parameter is “YES” which keeps the stored procedure process available for re-use at the next stored procedure call.

Note: There are also unfenced UDFs and Stored Procedures which run directly in an agent’s address space. By working this way, there is better performance. However, because they have unrestricted access to the agent’s address space, they need to be rigorously tested before being used.

Refer to the stored procedure chapter in the *Application Development Guide* for more information.

The multiple partition processing model is a logical extension of the single partition processing model. In fact, a single common code-base supports both modes of operation. The following figure is used to show the similarities and differences that exist between the single partition processing model as seen in the previous two figures, and the multiple partition processing model.

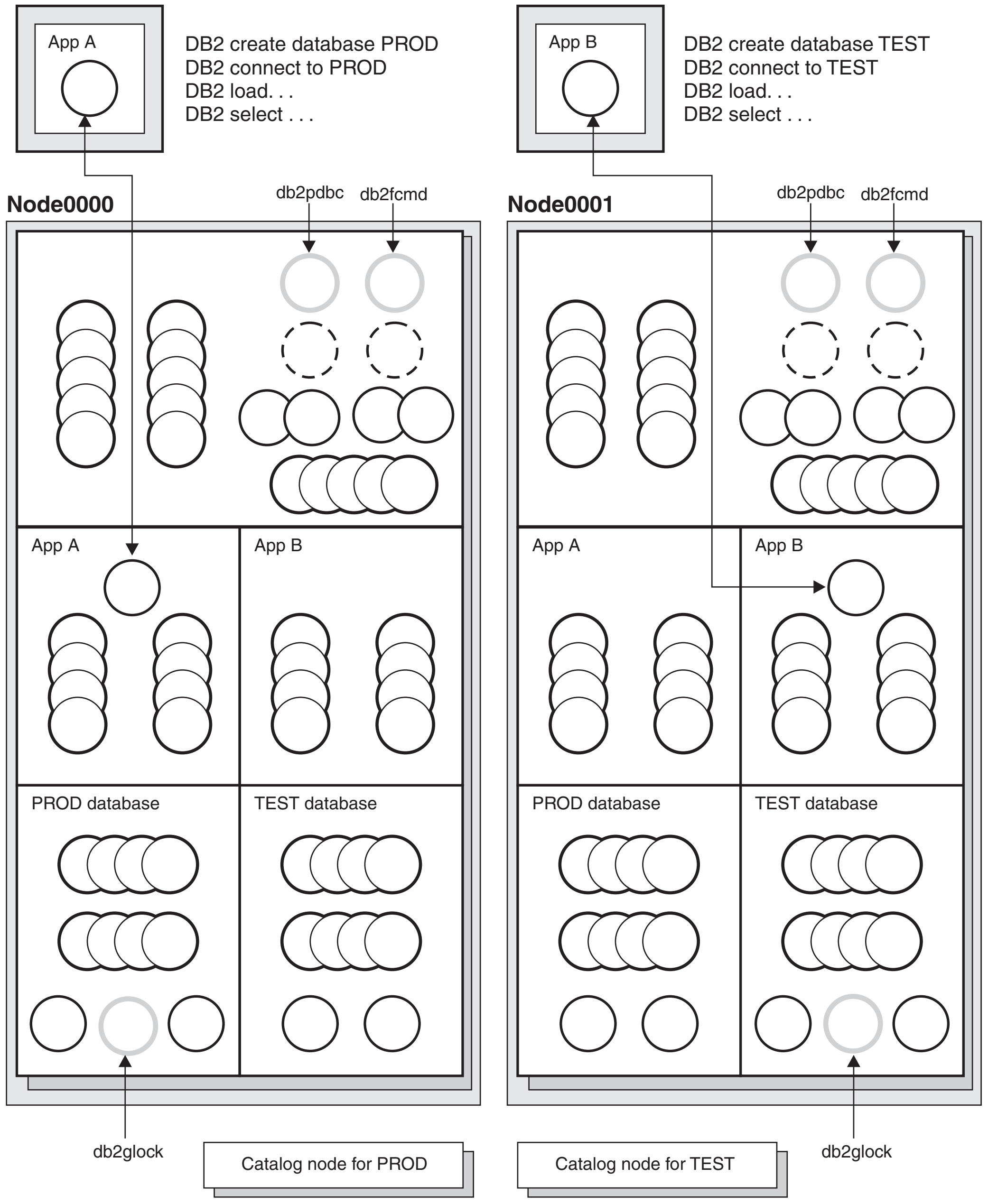


Figure 9. Process Model and Multiple Partitions

The majority of engine dispatchable units (EDUs) are the same between the single partition processing model and the multiple partition processing model.

In a multiple partition (or node) environment, one of the partitions must be considered the catalog node. The catalog keeps all of the information relating to the objects in the database.

As shown in the figure above, because Application A creates the PROD database on Node0000, the catalog for the PROD database is created on this node. Similarly, because Application B creates the TEST database on Node0001, the catalog for the TEST database is created on this node. You may wish to create your databases on different nodes because you will always want to balance the extra activity associated with the catalogs for each database across the nodes in your system environment.

There are additional EDUs (db2pdbc and db2fcmd) associated with the instance and these are found on each node in a multiple partition database environment. These EDUs are needed to coordinate requests across database partitions and to enable the Fast Communication Manager (FCM).

There is also an additional EDU (db2glock) associated with the catalog node for the database. This EDU controls global deadlock situations across the nodes where the active database is located.

Each CONNECT from an application is represented by a logical agent at the database and results in a single coordinator agent. The coordinator agent exists on the partition to which the application connected. This partition then becomes the “Coordinator Node” for that application. The coordinator node can also be set using the *SET CLIENT CONNECT_NODE* command. Parts of the database requests from the application are divided by the coordinator node to subagents at the other partitions; and all results from the other partitions are consolidated at the coordinator node before being sent back to the application.

The database partition where the CREATE DATABASE command was issued is called the “Catalog Node” for the database. It is at this database partition that the catalog tables are stored. Typically, all user tables are partitioned across a set of nodes.

Note: Any number of partitions can be configured to run on the same machine. This is known as a “Multiple Logical Partition”, or “Multiple Logical Node”, configuration. Such a configuration is very useful on large symmetric multiprocessor (SMP) machines with very large main memory. In this environment, communications between partitions can be optimized to use shared memory and semaphores.

Memory Model

Memory is important because it has a significant impact on how work gets done within the database. How you divide the available memory amongst those areas within the database is a primary way to control how well your database performs. You control this division of memory among the different heaps through configuration parameters. We will present the key configuration parameters and what part of memory they control in this section. See "How DB2 Uses Memory" on page 227 for more information on this topic.

All engine dispatchable units (EDUs) in a partition are attached to the Instance Shared Memory. All EDUs doing work within a database are attached to that database's Database Shared Memory. All EDUs working on behalf of a particular application are attached to an Application Shared Memory region for that application. This type of shared memory is only allocated if intra- or inter-partition parallelism is enabled. Finally, each EDU has its own private memory.

Instance Shared Memory (also known as Database Manager Shared Memory) is allocated when the database is started. From the Instance Shared Memory all other memory is attached/allocated. If Fast Communication Manager (FCM) is used there are buffers taken from this memory. FCM is used for internal communications, primarily messages, both among and within the database servers in a particular database environment. When the first application connects or attaches to a database, Database Shared, Application Shared, and Agent Private memory areas are allocated.

Database Shared Memory (also known as Database Global Memory) is allocated when a database is activated or connected to for the first time. This memory is used across all applications that might connect to the database. Many different memory areas are contained in database shared memory including:

- Buffer pools
- Lock list
- Database heap – and this includes the log buffer and the catalog cache.
- Utility heap
- Package cache

The database manager configuration parameter *numdb* specifies the number of local databases that can be concurrently active. In a partitioned database environment, this parameter limits the number of active database partitions on a database partition server. The value of the *numdb* parameter may impact the total amount of memory allocated.

Application Shared memory (also known as Application Global Memory) is allocated when an application connects to a database. This allocation occurs only in a partitioned database environment, or if the database manager configuration parameter *intra_parallel* is enabled. This memory is used by agents working on behalf of the application to share data and coordinate activities amongst themselves.

The database configuration parameter *maxappls* sets an upper limit to the number of applications that connect to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

To a certain extent, the maximum number of applications is also governed by the database manager configuration parameter *maxagents* (or *max_coordagents* for parallel environments). The *maxagents* parameter sets an upper limit to the total number of database manager agents in a partition. These database manager agents include active coordinator agents, subagents, inactive agents, and idle agents.

Agent Private Memory is allocated for an agent when that agent is assigned to work for a particular application. The agent private memory is allocated for the agent and contains memory allocations that will be used only by this specific agent, such as the sort heap and the application heap.

There are a few special types of shared memory:

- Agent/Local Application Shared Memory. This memory is used for SQL request and response communications between an agent and its client application.
- UDF/Agent Shared Memory. This memory is attached to by agents running a fenced UDF or Stored Procedure. It is used as a communications area.
- Extended Storage. A typically very large (greater than 4 GB) region of shared memory used as an extended buffer pool. Agents/Prefetchers/Page cleaners are not permanently attached to it, but attach to individual segments within it as needed.

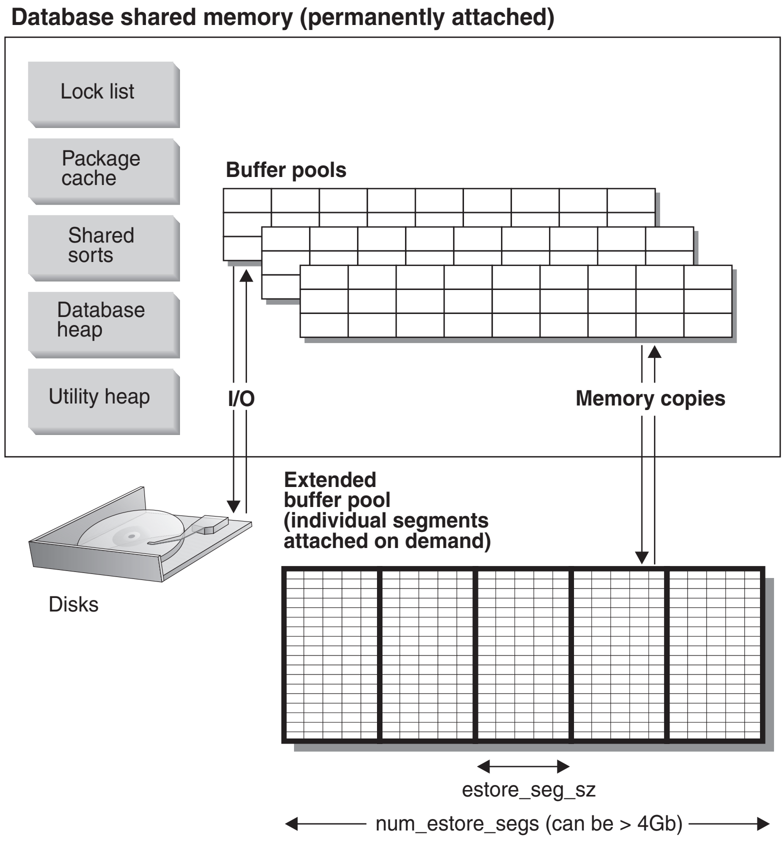

Database shared memory (permanently attached)
Lock list
Package cache
Shared sorts
Database heap
Utility heap
Buffer pools
I/O
Memory copies
Extended buffer pool (individual segments attached on demand)
Disks
estore_seg_sz
num_estore_segs (can be > 4Gb)

Figure 10. Buffer Pools and Extended Storage

Extended storage acts as an extended look-a-side buffer for the main buffer pool(s). See “Extending Memory” on page 265 for more information on this topic. It can be much larger than 4 GB and is an excellent way to exploit machines with large amounts of main memory. The extended storage cache is defined in terms of memory segments.

You should be aware when deciding to use some of the real addressable memory as an extended storage cache that this memory can then no longer be used of other purposes on the machine such as a jfs-cache or as process private address space. Assigning additional real addressable memory to the extended storage cache could lead to higher system paging.

The following database configuration parameters influence the amount and size of the memory available for extended storage:

- *num_estore_segs* defines the number of extended storage memory segments.
- *estore_seg_sz* defines the size of each extended memory segment.

Each table space is assigned a buffer pool. An extended storage cache must always be associated with one or more specific buffer pools. The page size of the extended storage cache must match the page size of the buffer pool it is associated with.

See “Extending Memory” on page 265 for more information on the extended storage cache.

Part 2. Tuning Application Performance

Chapter 3. Application Considerations

There are a number of factors that can impact the runtime performance of your application. This chapter describes the following topics that should be considered when you are designing and coding your application:

- Concurrency
- Locking
- Adjusting the Optimization Class
- Restrictions on Result Sets to Improve Performance
- Row Blocking
- Tuning Queries
- Compound SQL
- Performance Considerations and Character Conversion
- Stored Procedures
- Activating a Database
- Parallel Processing of Applications.

You should also refer to the *Application Development Guide* and the *CLI Guide and Reference* for additional information which can affect the performance of your applications, for example:

- Writing programs using embedded static SQL
- Writing programs using embedded dynamic SQL
- Writing programs using DB2 Call Level Interface (CLI).

Concurrency

The integrity of the data in a relational database must be maintained as multiple users access and change the data. *Concurrency* is the sharing of resources by multiple interactive users or application programs at the same time. The database manager controls this access to prevent undesirable effects, such as:

- *Lost updates*. Two applications, A and B, might both read the same row from the database and both calculate new values for one of its columns based on the data these applications read. If A updates the row with its new value and B then also updates the row, the update performed by A is lost.
- *Access to uncommitted data*. Application A might update a value in the database, and application B might read that value before it was committed.

Then, if the value of A is not later committed, but backed out, the calculations performed by B are based on uncommitted (and presumably invalid) data.

- *Nonrepeatable reads*. Some applications involve the following sequence of events: application A reads a row from the database, then goes on to process other SQL requests. In the meantime, application B either modifies or deletes the row and commits the change. Later, if application A attempts to read the original row again, it receives the modified row or discovers that the original row has been deleted.
- *Phantom Read Phenomenon*. The phantom read phenomenon occurs when:
  1. Your application executes a query that reads a set of rows based on some search criterion.
  2. Another application inserts new data or updates existing data that would satisfy your application's query.
  3. Your application repeats the query from step 1 (within the same unit of work).

  When the query is repeated (step 3), some additional ("phantom") rows are returned as part of the result set that were not returned when the query was initially executed (step 1).

An *isolation level* determines how data is locked or isolated from other processes while the data is being accessed. The isolation level will be in effect for the duration of the unit of work. Applications that use a cursor declared using the WITH HOLD clause will keep the chosen isolation level for the duration of the unit of work in which the OPEN CURSOR was performed. (For more information, refer to the *SQL Reference* manual.) See "Specifying the Isolation Level" on page 46 for information on how the isolation level is specified.

DB2 supports the following isolation levels:
- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read.

(Note that some DRDA database servers support the *no commit* isolation level. On other databases, it behaves like the uncommitted read isolation level. Refer to the *SQL Reference* for information on this isolation level.)

See also:
- "Choosing the Isolation Level" on page 46
- "Specifying the Isolation Level" on page 46.

It may be that you are working in a `federated database system` that supports applications and users submitting SQL statements referencing two or more database management systems (DBMSs) or databases in a single statement. A DB2 federated system provides `location transparency` for database objects. For example, if information about tables and views is moved, references to that information (called `nicknames`) can be updated without changes to applications that request the information. When an application accesses nicknames, DB2 relies on the concurrency control protocols of data source database managers to ensure isolation levels. (A `data source` consists of a DBMS and data.) DB2 will attempt to match the requested level of isolation at the data source with a logical equivalent; however, results may vary based on data source capabilities. Refer to the *Application Development Guide* manual for information on writing applications accessing nicknames.

Detailed explanations for each of the isolation levels follows in decreasing order of performance impact, but in increasing order of care required when accessing and updating data.

Repeatable Read

Repeatable read (RR) locks all the rows an application references within a unit of work. Using repeatable read, a SELECT statement issued by an application twice within the same unit of work in which the cursor was opened, gives the same result each time. With repeatable read, lost updates, access to uncommitted data, and phantom rows are not possible.

The repeatable read application can retrieve and operate on the rows as many times as needed until the unit of work completes. However, no other applications can update, delete, or insert a row that would affect the result table, until the unit of work completes. Repeatable read applications cannot see uncommitted changes of other applications.

With repeatable read, every row that is referenced is locked, not just the rows that are retrieved. Appropriate locking is performed so that another application cannot insert or update a row that would be added to the list of rows referenced by your query, if the query was re-executed. This prevents phantom rows from occurring. This means that if you scan 10 000 rows and apply predicates to them, locks are held on all 10 000 rows, even though only 10 rows qualify.

Note: The repeatable read isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

Since repeatable read may acquire and hold a considerable number of locks, these locks may exceed the number of locks available as a result of the *locklist* and *maxlocks* configuration parameters. (See “Maximum Percent of Lock List

Before Escalation (maxlocks)” on page 364 and “Maximum Storage for Lock List (locklist)” on page 335.) In order to avoid lock escalation, the optimizer may elect to immediately acquire a single table level lock for an index scan, if it believes that lock escalation is very likely to occur. (See “Lock Escalation” on page 53 for a discussion of lock escalation.) This functions as though the database manager has issued a LOCK TABLE statement on your behalf. If you do not want a table level lock to be obtained ensure that enough locks are available to the transaction or use the Read Stability isolation level.

Read Stability

Read stability (RS) locks only those rows that an application retrieves within a unit of work. It ensures that any qualifying row read during a unit of work is not changed by other application processes until the unit of work completes, and that any row changed by another application process is not read until the change is committed by that process. That is, “nonrepeatable read” behavior is **not** possible.

Unlike repeatable read, with read stability, if your application issues the same query more than once, you may see additional *phantom* rows (the *phantom read phenomenon*). Recalling the example of scanning 10 000 rows, read stability only locks the rows that qualify. Thus, with read stability, only 10 rows are retrieved, and a lock is held only on those ten rows. Contrast this with repeatable read, where in this example, locks would be held on all 10 000 rows. The locks that are held can be share, next share, update, or exclusive locks. (For more information on lock attributes, see “Attributes of Locks” on page 49.)

Note: The read stability isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

One of the objectives of the read stability isolation level is to provide both a high degree of concurrency as well as a stable view of the data. To assist in achieving this objective, the optimizer ensures that table level locks are not obtained until lock escalation occurs. (See “Lock Escalation” on page 53 for more information about lock escalation).

The read stability isolation level is best for applications that include all of the following:
- Operate in a concurrent environment
- Require qualifying rows to remain stable for the duration of the unit of work
- Do not issue the same query more than once within the unit of work, or do not require that the query get the same answer when issued more than once in the same unit of work.

Cursor Stability

Cursor stability (CS) locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.

No other applications can update or delete a row that a cursor stability application has retrieved while any updatable cursor is positioned on the row. Cursor stability applications cannot see uncommitted changes of other applications.

Recalling the example of scanning 10 000 rows, if you use cursor stability, you will only have a lock on the row under your current cursor position. The lock is removed when you move off that row (unless you update that row).

With cursor stability, both nonrepeatable read and the phantom read phenomenon are possible. Cursor stability is the default isolation level and should be used when you want the maximum concurrency while seeing only committed rows from other applications.

Uncommitted Read

Uncommitted read (UR) allows an application to access uncommitted changes of other transactions. The application also does not lock other applications out of the row it is reading, unless the other application attempts to drop or alter the table. Uncommitted read works differently for read-only and updatable cursors.

Read-only cursors can access most uncommitted changes of other transactions. However, tables, views, and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes by other transactions can be read before they are committed or rolled back.

Note: Cursors that are updatable operating under the uncommitted read isolation level will behave as if the isolation level was cursor stability.

Recalling the example of scanning 10 000 rows, if you use uncommitted read, you do not acquire any row locks.

With uncommitted read, both nonrepeatable read behavior and the phantom read phenomenon are possible.

The uncommitted read isolation level is most commonly used for queries on read-only tables, or if you are only executing select-statements and you do not care whether you see uncommitted data from other applications.

Choosing the Isolation Level

Table 1 summarizes the different isolation levels in terms of the undesirable effects described in *Application Development Guide* manual.

Table 1. Summary of isolation levels

Isolation Level	Access to Uncommitted Data	Nonrepeatable Reads	Phantom Read Phenomenon
Repeatable Read (RR)	Not Possible	Not Possible	Not Possible
Read Stability (RS)	Not Possible	Not Possible	Possible
Cursor Stability (CS)	Not Possible	Possible	Possible
Uncommitted Read (UR)	Possible	Possible	Possible

Table 2 provides a simple heuristic that may help you choose an initial isolation level for your applications. Consider this table as a starting point, and refer to the previous discussions of the various levels for factors that might make another value more appropriate for your requirements.

Table 2. Guidelines for choosing an isolation level

Application Type	High data stability required	High data stability not required
Read-write transactions	RS	CS
Read-only transactions	RR or RS	UR

Choosing the appropriate isolation level for an application is very important to avoid the phenomena that are intolerable for that application. The isolation level affects not only the degree of isolation among applications but also the performance characteristics of an individual application since the CPU and memory resources, required to obtain and free locks, vary with the isolation level. The potential for deadlock situations also varies with the isolation level.

Specifying the Isolation Level

The isolation level is specified at precompile time or when an application is bound to a database. For an application written in a supported compiled language, use the ISOLATION option of the command line processor PREP or BIND commands. The isolation level can also be specified by using the PREP or BIND APIs. If no isolation level is specified, the default of cursor stability is used.

If a bind file is created at precompile time, the isolation level is stored in the bind file. If no isolation level is specified at bind time, the default is the isolation level used during precompilation.

You can determine the isolation level of a package by executing the following query:

```
SELECT ISOLATION FROM SYSCAT.PACKAGES
  WHERE PKGNAME = 'XXXXXXXX'
  AND PKGSCHEMA = 'YYYYYYYY'
```

where `XXXXXXXX` is the name of the package and `YYYYYYYY` is the schema name of the package. Both of these names must be in all capital letters.

When a database is created, multiple bind files used to support the different isolation levels for SQL in REXX are bound to the database (on those servers that support REXX). Other command line processor packages are also bound to the database when a database is created. Refer to the *Application Development Guide* for more information about bind files.

REXX and the command line processor connect to a database using a default isolation level of cursor stability. Changing to a different isolation level does not change the connection state. It must be executed in the CONNECTABLE AND UNCONNECTED state or in the IMPLICITLY CONNECTABLE state. (Refer to the CONNECT TO statement in the *SQL Reference* for details about connection states.) You cannot be connected to a database when issuing this command.

The isolation level being used can be checked by a REXX application by checking the value of the SQLISL REXX variable. The value is updated every time the CHANGE SQLISL command is executed.

The DB2_RR_TO_RS profile registry variable can be used to prevent Repeatable Read (RR) isolation level access to user tables. This registry value can be set to "YES" using `db2set` in environments where RR isolation semantics are not required. Before taking effect, you must stop and start the database. Following the `db2start`, this change affects the entire instance. Once set, if a request to access a user table using RR is received, the request is modified internally to use the Read Stability (RS) isolation level instead. No warning is given when this occurs.

If you are using the command line processor you may change the isolation level using the CHANGE ISOLATION LEVEL command. Refer to the *Command Reference* manual for more information.

For DB2 Call Level Interface (DB2 CLI), you may change the isolation level as part of the DB2 CLI configuration. Within CLI at runtime, you would use the SQLSetConnectAttr function with the SQL_ATTR_TXN_ISOLATION attribute. This will set the transaction isolation level for the current connection referenced by the *ConnectionHandle*. Within the db2cli.ini file you could also use the TXNISOLATION keyword.

Note: JDBC and SQLJ are implemented with CLI on DB2, which means the db2cli.ini settings may affect what is written and run using JDBC and SQLJ. Refer to the *CLI Guide and Reference* manual for more information.

When working with JDBC or SQLJ at runtime, you can use the setTransactionIsolation method within the java.sql interface connection to establish the isolation level. Refer to the “Programming in Java” chapter of the *Application Development Guide* manual for more information.

When working with SQLJ, if you run the db2profc SQLJ optimizer, a package is created. The final options for this package may be specified to include the isolation level to be used. Refer to the “Programming in Java” chapter of the *Application Development Guide* manual for more information.

In addition, many commercially-written applications also provide a method to allow you to choose the isolation level. Refer to the *CLI Guide and Reference* manual for more information.

Declared Temporary Tables and Concurrency

Declared temporary tables have no concurrency issues since they are only available to the application that declared them. This type of table only exists from the time that the application declares it until the application completes or disconnects.

Locking

The database manager provides concurrency control and prevents uncontrolled access by means of locks. A *lock* is a means of associating a database manager resource with an application to control how other applications can access the same resource. The application with which the resource is associated is said to hold or own the lock.

The database manager imposes locks to prohibit applications from accessing uncommitted data written by other applications (unless the uncommitted read isolation level is used). This principle protects data integrity (that is, the consistency and security of data). Locks can also prohibit the updating of rows (such as for a repeatable read application).

To satisfy data integrity, the database manager acquires locks implicitly, under database manager control. Except for the uncommitted read isolation level, it is never necessary for an application to request a lock explicitly to ensure that uncommitted data is hidden from other processes.

Because of the basic principle of locking, you do not need to take action to control locks in most cases. Still, applications acquire locks on the basis of certain general parameters. Knowledge of your local situation can help you

make better use of your system resources by changing those parameters. To assist you, the following topics on locking are discussed:
- Attributes of Locks
- Locks and Application Performance
- Factors Affecting Locking
- LOCK TABLE Statement
- CLOSE CURSOR WITH RELEASE
- Summary of Locking Considerations.

Attributes of Locks

Database manager locks have the following basic attributes:

Mode The type of access allowed for the lock owner as well as the type of access permitted for concurrent users of the locked object. It is sometimes referred to as the *state* of the lock.

Object
The resource being locked. The only types of explicitly lockable objects are tables. The database manager also imposes locks on other types of resources, such as rows, tables and table spaces. The object being locked represents the *granularity* of the lock.

Duration
The length of time a lock is held. Lock durations are affected by isolation levels which are discussed in "Concurrency" on page 41.

In the following table, modes and their effects are shown in order of increasing control over resources:

Table 3. Lock Mode Summary

Lock Mode	Applicable Object Type	Description
IN (Intent None)	Table spaces, tables	The lock owner can read any data in the table, including uncommitted data, but cannot update any of it. No row locks are acquired by the lock owner. Other concurrent applications can read or update the table.
IS (Intent Share)	Table spaces, tables	The lock owner can read data in the locked table, but not update this data. When an application holds the IS table lock, the application acquires an S or NS lock on each row read. In either case, other applications can read or update the table.
NS (Next Key Share)	Rows	The lock owner and all concurrent applications can read, but not update, the locked row. This lock is acquired on rows of a table, instead of an S lock, where the isolation level is either RS or CS on data that is read.

Table 3. Lock Mode Summary (continued)

Lock Mode	Applicable Object Type	Description
S (Share)	Rows, tables	The lock owner and all concurrent applications can read, but not update, the locked data. Individual rows of a table can be S locked. If a table is S locked, no row locks are necessary.
IX (Intent Exclusive)	Table spaces, tables	The lock owner and concurrent applications can read and update data in the table. When the lock owner reads data, an S, NS, X, or U lock is acquired on each row read. An X lock is also acquired on each row that the lock owner updates. Other concurrent applications can both read and update the table.
SIX (Share with Intent Exclusive)	Tables	The lock owner can read and update data in the table. The lock owner acquires X locks on the rows it updates, but acquires no locks on rows that it reads. Other concurrent applications can read the table.
U (Update)	Rows, tables	The lock owner can update data in the locked row or table. The lock owner acquires X locks on the rows before it updates the rows. Other units of work can read the data in the locked row or table; but cannot attempt to update it.
NX (Next Key Exclusive)	Rows	The lock owner can read but not update the locked row. This mode is similar to an X lock except that it is compatible with the NS lock.
NW (Next Key Weak Exclusive)	Rows	This lock is acquired on the next row when a row is inserted into the index of a non-catalog table. The lock owner can read but not update the locked row. This mode is similar to X and NX locks except that it is compatible with the W and NS locks.
X (Exclusive)	Rows, tables	The lock owner can both read and update data in the locked row or table. Tables can be Exclusive locked, meaning that no row locks are acquired on rows in those tables. Only uncommitted read applications can access the locked table.
W (Weak Exclusive)	Rows	This lock is acquired on the row when a row is inserted into a non-catalog table. The lock owner can change the locked row. This lock is similar to an X lock except that it is compatible with the NW lock. Only uncommitted read applications can access the locked row.
Z (Superxclusive)	Table spaces, tables	This lock is acquired on a table in certain conditions, such as when the table is altered or dropped, an index on the table is created or dropped, or a table is reorganized. No other concurrent application can read or update the table.

Note: Only tables and table spaces will obtain the “intent” lock modes. That is, intent locks are not obtained for rows.

Locks and Application Performance

Application programmers need to be aware of several related factors concerning the uses of locks and their effect on the performance of applications. These factors include the following:

- Concurrency and Granularity
- Lock Compatibility
- Lock Conversion
- Lock Escalation
- Lock Waits and Timeouts
- Deadlocks.

Concurrency and Granularity

A lock held by one application can prevent access by another application. Therefore, for maximum concurrency, a row level lock is better than a table lock. But locks require storage and processing time to manage. Therefore, for minimizing storage and processing time, a single table lock is better than many row locks.

You can define the size (granularity) of locks at row or table level through the LOCKSIZE clause of the ALTER TABLE statement. By default, row locks are used. With permanent table locks, as defined by ALTER TABLE, only S and X table locks are used. Performance is improved since the application does not need to acquire and release as many row locks. You may prefer to get a permanent table lock using the ALTER TABLE statement rather than a single transaction table lock using LOCK TABLE statement in the following cases:

- Your table is read-only, and you will always need S locks. A table level lock will improve performance while allowing others to obtain S locks on the table.
- The table will be accessed by a single user for maintenance, where the person requires an X lock, for a limited period of time. Changing a table level lock through ALTER TABLE on the table, will provide the person with an X lock at a table level. Once the person is finished, they can use ALTER TABLE to return the table to row level locking.

Use of the ALTER TABLE statement will not prevent normal lock escalation from occurring.

In addition, note that using ALTER TABLE to push locks to the table level is a global approach, affecting all applications and users that access that table. Another choice is for individual applications to use the LOCK TABLE statement. This allows you to go to table locks at an application level, not a database level (as mentioned in the second bullet above.

Lock Compatibility

Table 4 indicates whether a lock request is granted if another process holds or is requesting a lock on the same resource in a given state. A **no** indicates that the requestor must wait until all incompatible locks are released by other processes. Note that a timeout can occur when waiting for a lock. A **yes** indicates that the lock is granted (unless someone else is waiting for the resource).

Table 4. Lock Type Compatibility

	State of Held Resource												
State Being Requested	**none**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**NX**	**X**	**Z**	**NW**	**W**
none	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IN	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
IS	yes	yes	yes	yes	yes	yes	yes	yes	no	no	no	no	no
NS	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes	no
S	yes	yes	yes	yes	yes	no	no	yes	no	no	no	no	no
IX	yes	yes	yes	no	no	yes	no	no	no	no	no	no	no
SIX	yes	yes	yes	no	no	no	no	no	no	no	no	no	no
U	yes	yes	yes	yes	yes	no	no	no	no	no	no	no	no
NX	yes	yes	no	yes	no	no	no	no	no	no	no	no	no
X	yes	yes	no	no	no	no	no	no	no	no	no	no	no
Z	yes	no	no	no	no	no	no	no	no	no	no	no	no
NW	yes	yes	no	yes	no	no	no	no	no	no	no	no	yes
W	yes	yes	no	no	no	no	no	no	no	no	no	yes	no

Note:

I	Intent
N	None
NS	Next Key Share
S	Share
NX	Next Key Exclusive
X	Exclusive
U	Update
Z	Super Exclusive
NW	Next Key Weak Exclusive
W	Weak Exclusive

For details of these lock types, refer to the discussion in “Attributes of Locks” on page 49.

Note:
- yes - **grant** lock requested immediately
- no - **wait** for held lock to be released or timeout to occur

Assume that application A holds a lock on a table that application B also wants to access. The database manager requests, on behalf of application B, a lock of some particular mode. If the mode of the lock held by A permits the lock requested by B, the two locks (or modes) are said to be compatible.

If the lock mode requested for application B is not compatible with the lock held by application A, application B cannot continue. Instead, it must wait until application A releases its lock, and *all* other existing incompatible locks are released.

Lock Conversion

Lock conversion occurs when a process accesses a data object on which it already holds a lock, and the mode of access requires a more restrictive lock than the one already held. A process can hold only one lock on a data object at any time, although it can (indirectly through a query) request a lock many times on the same data object. The operation of changing the mode of the lock already held is called a *conversion*.

The conversion case for rows is simple: As an example, a conversion occurs if an X is needed and an S or U is held.

There are distinct lock modes for tables and for rows. IX (Intent Exclusive) and S (Shared) locks are special cases with regard to lock conversion, however. Neither S nor IX is considered to be more restrictive than the other, so if one of these is held and the other required, the resulting conversion is to a SIX (Share with Intent Exclusive) lock. All other conversions result in the requested lock mode becoming the mode of the lock held, if the requested mode is less restrictive.

A query to update a row can also produce a dual conversion. Suppose the row had been read through an index access and was locked as S. The table containing the row would have a covering intention lock. Suppose it is an IS rather than an IX. Then, if the row is subsequently changed, the table lock is converted to an IX, and the row to an X.

As a reminder, the application of locks usually takes place implicitly during the execution of a query. Understanding the kinds of locks obtained for different queries and table and index combinations can assist you in designing and tuning your application. See “Factors Affecting Locking” on page 57 for more information on this topic.

Lock Escalation

Lock escalation is an internal mechanism to reduce the number of locks held. Escalation is from many row locks (in a single table) to a single table lock.

Lock escalation occurs when too many locks (of any type) are currently held.

Lock escalation can occur for a specific database agent if the agent exceeds its allocation of the lock list (see “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 364).

Such escalation is handled internally; the only externally detectable result might be a reduction in concurrent access on one or more tables. Normally, in a properly configured database, lock escalation occurs infrequently.

Lock escalation can occur when, for example, an application designer uses an index on a large table to increase performance and concurrency; however, the application accesses a large percentage of records in the table. The database manager is not able to predict (in this case) that so much of the table will be locked, and locks each record individually rather than only locking the table either S or X. As a solution to this case, and after consulting with the application designer, the database designer could make a recommendation to use a LOCK TABLE statement for this transaction.

Sometimes, the process receiving the escalation request (internally) holds few or no record locks on any table. The reason for this escalation is that one process (or processes) can be holding many locks (although this amount is below the database configuration parameter of locks per process) but not quite enough to trigger the escalation request. The process might not request another lock or access the database again except to end the transaction. Then another process can request the lock or locks that trigger the escalation request.

If lock escalation reduces concurrency to an unacceptable level, you can do the following:

- Check the contents of the *db2diag.log* for information on escalations. Information is recorded for each table being escalated. The type of information recorded includes:
  - The number of locks currently held.
  - The number of locks needed before lock escalation is completed.
  - The table identifier information and table name of each table being escalated.
  - The number of non-table locks currently held.
  - The new table level lock to be acquired as part of the escalation. Typically, this will be a “S” or Share lock, or an “X” or eXclusive lock.
  - The internal return code of the result of the acquisition of the new table lock level.

  The current dynamic SQL statement may also be recorded. If it is, the information recorded will include the current SQL statement prior to the escalation of any table locks **if** the DIAGLEVEL database manager configuration parameter is 4. If lock escalation fails, the information recorded will include the table for which the escalation failed and the current SQL statement (if it is available, and not previously written) **if** the DIAGLEVEL is 2 or higher.

With this information you will be able to carry out an appropriate action based on the other points mentioned below.

To start this type of information recording, you should set the database manager configuration parameter DIAGLEVEL to 3 which is the default, or to 4.

- Increase the number of locks allowed by increasing the value of the *maxlocks* and/or the *locklist* parameters in the database configuration file. (See “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 364 and “Maximum Storage for Lock List (locklist)” on page 335.) This might be the choice if concurrent access to the table by other processes is most important. However, the overhead of obtaining record level locks can induce more delay to other processes than is saved by concurrent access to a table. (When changing these parameters in a partitioned database, ensure that the parameters are updated on all partitions).
- Locate and adjust the offending process (or processes), which may or may not be the one escalating or rolling back, and issue LOCK TABLE statements explicitly.
- Change the degree of isolation. Note that this may lead to decreased concurrency.
- Increase the frequency of commits. This tends to reduce the number of locks in existence at a given time. For more information about isolation levels and concurrency, see “Concurrency” on page 41.

Lock Waits and Timeouts

Without lock timeout detection, in an abnormal situation, your application may have to wait indefinitely for a lock to be released. This might occur, for example, when a transaction is waiting for a lock held by another user’s application, and the other user has left their workstation without performing some interaction to allow their application to commit their transaction which would release the lock. Obviously, this results in poorer application performance. To avoid *stalling* your program in such a case, you can use the *locktimeout* configuration parameter to set the maximum time that any application waits to obtain a lock. (See “Lock Timeout (locktimeout)” on page 365.)

Using this parameter helps avoid global deadlocks, especially in distributed unit of work (DUOW) applications. If the lock times out, that is, if the time that the lock request is pending is greater than the *locktimeout* value, your application receives an error and your transaction is rolled back. For example, if *program1* tries to acquire a lock which is already held by *program2*, *program1* returns SQLCODE -911 (SQLSTATE 40001) with reason code 68 if the timeout is expired.

If the database manager configuration parameter *diaglevel* is set to four, and a lock request times-out, more information can be found in the db2diag.log. The information found there includes the object, the lock mode, and the application holding the lock on the object. The current dynamic SQL statement or static package name may also be found.

Deadlocks

In the database manager, contention for locks by processes using the database can result in deadlocks. For example, Process 1 locks table A in X (exclusive) mode and Process 2 locks table B in X mode; if Process 1 then tries to lock table B in X mode and Process 2 tries to lock table A in X mode, the processes will be in a deadlock. In a deadlock, both processes are suspended until their second lock request is granted, and neither request is granted until one of the processes performs a commit or rollback. This state remains indefinitely until an external agent activates one of the processes and forces it to perform a rollback.

Deadlocks in the lock system are handled in the database manager by an asynchronous system background process called the deadlock detector. The deadlock detector becomes active periodically as determined by the *dlchktime* configuration parameter (see “Time Interval for Checking Deadlock (dlchktime)” on page 363). When the deadlock detector becomes active, it examines the lock system for deadlocks. If the database has been partitioned then each partition sends *lock graphs* to the database partition having the system catalog views which is where global deadlock detection takes place.

If a deadlock is found, the deadlock detector selects a deadlocked process to roll back. The selected process is awakened, and it returns to the calling application with SQLCODE -911 (SQLSTATE 40001), with reason code 2. The database manager rolls back the selected process automatically. When the rollback has completed, the locks belonging to the victim process are released, and the other processes involved in the deadlock can eventually proceed.

Selecting the proper interval for the deadlock detector is necessary to ensure good performance. An interval that is too short would cause unnecessary overhead, and one that is too long would allow a deadlock to delay a process for an unacceptable amount of time. For example, a wake up interval set to 30 minutes could allow a deadlock to exist for nearly 30 minutes. The application designer must balance the possible delays in resolving deadlocks with the overhead of detecting them.

In a partitioned database, the interval should be the same on all partitions (the *dlchktime* configuration parameter must be updated to the same value on all partitions). If the value is smaller at the catalog node than at other partitions, phantom deadlocks may be detected. If the value is larger at the catalog node than at other partitions, it may appear as if more than two intervals pass before a deadlock is detected. If a large number of deadlocks are detected in a partitioned database, you should increase the value of the *dlchktime* parameter to account for lock waits and communication waits.

Another problem can occur when an application with more than one independent process accessing the database is structured in such a way as to make deadlocks likely. An example is an application in which several processes access the same table for reads and then writes. If the processes do read-only SQL queries at first and then do SQL updates on the same table, the chances of deadlocks occurring increase because of potential contention between the processes for the same data. For instance, if two processes read the table, and then update the table, they get into a state where process A is trying to get an X lock on a row, on which process B has an S lock and vice versa. The result could be a deadlock. To avoid these deadlocks, applications that access data with the intention of modifying it should use the FOR UPDATE OF clause when performing a select. This clause ensures that a U lock is imposed when process A attempts to read the data.

Note: You may want to consider defining a monitor that will record when deadlocks occur. Use the CREATE EVENT statement described in the *SQL Reference* to create the monitor.

In a federated system environment, when an application accesses nicknames, it is possible that the data requested by the application is unavailable due to a deadlock at a data source. When this happens, DB2 relies on the deadlock handling facilities at the data source to resolve the lock. In the case of deadlocks across more than one data source, DB2 relies on data source timeout mechanisms to break the deadlock.

If the database manager configuration parameter *diaglevel* is set to four, and a lock request fails because of a deadlock, more information can be found in the db2diag.log. The information found there includes the object, the lock mode, and the application holding the lock on the object. The current dynamic SQL statement or static package name may also be found.

Factors Affecting Locking

The mode and granularity of database manager locks are determined by a combination of factors: the type of processing the application performs, how it accesses data, and several parameters that you can specify.

Application Processing

For the purpose of determining lock attributes, processing can be classified as one of four types:

Read-only
: This type includes all select-statements which are intrinsically read-only (refer to the *SQL Reference* for information about cursors), have an explicit FOR READ ONLY clause, or are ambiguous but for which the SQL compiler presumes to be read-only due to the value of the BLOCKING option specified on the PREP or BIND command. It requires only Share locks (S or IS).

Intent to change
: This type includes all select-statements with the FOR UPDATE clause, or which the SQL compiler presumes to be intended for change as a result of the interpretation of the ambiguous statement. It uses Share and Update locks (S, U, and X for rows, IX, U, X for tables).

Change
: This type includes UPDATE, INSERT, and DELETE, but not UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. It requires Exclusive locks (X or IX).

Cursor controlled
: This type includes UPDATE WHERE CURRENT OF and DELETE WHERE CURRENT OF. It also requires Exclusive locks (X or IX).

A statement that inserts, updates or deletes against a target table, based on the result from a sub-select statement, does two types of processing. The locks for the tables returned in the sub-select are determined by the rules for read-only processing; for the target table, by the rules for change processing.

Access Paths

An *access path* is the method selected by the optimizer for retrieving data from a specific table reference. (See “Data Access Concepts and Optimization” on page 153.) The access path chosen by the optimizer can have a significant effect on the lock modes. For example, when an index scan is used to locate a specific row, the optimizer will likely choose row-level locking (IS) for the table. This type of access would be used to select information for a single employee from the EMPLOYEE table, that has an index on employee number (EMPNO), with a statement such as the following:

```
SELECT *
  FROM EMPLOYEE
  WHERE EMPNO = '000310';
```

Similarly, when no index is used, the entire table must be scanned in sequence to find the selected rows, and may acquire a single table level lock (S). For example, this type of access might be used to select all the male employees, using a statement such as this where there is no index on the column SEX:

```
SELECT *
  FROM EMPLOYEE
  WHERE SEX = 'M';
```

The following tables provide an overview of which locks are obtained for what kind of access plan. See “Application Processing” on page 58 for definitions of the column headings. Also see “Data Access Concepts and Optimization” on page 153 for definitions of the access method. Note that *cursor controlled* type processing uses the lock mode of the underlying cursor until the application finds a row to update or delete. For this type of processing, no matter what the lock mode of a cursor, an exclusive lock will always be obtained to perform the update or delete.

In the following tables, if only one lock mode is shown, it is a table level lock mode. If two lock modes are shown, the first is the table level lock mode and the second is the row level lock mode.

Table 5. Lock Modes for Table Scans

Isolation Level	Read-only	Intent to Change	Change
Access Method: *Table scan with no predicates*			
RR	S	U	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Table Scan with predicates*			
RR	S	U	U
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

Table 6. Lock Modes for Index Scans

Isolation Level	Read-only	Intent to Change	Change
Access Method: *Index Scan with no predicates*			
RR	S	IX / U	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Index Scan a single qualifying row*			

Table 6. Lock Modes for Index Scans (continued)

Isolation Level	Read-only	Intent to Change	Change
RR	IS / S	IX / U	IX / X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Index Scan with start and stop predicates only*			
RR	IS / S	IX / S	IX / X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Index Scan with predicates*			
RR	IS / S	IX / S	IX / U
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

Table 7 shows the lock modes for cases in which reading of the data pages is deferred to allow the list of rows to be:
- Further qualified using multiple indexes. See "Multiple Index Access" on page 159 for more information.
- Sorted for efficient prefetching. See "Understanding List Prefetching" on page 243 for more information.

The deferred access of the data pages implies that access to the row occurs in two steps and this results in more complex locking scenarios. There are two major categories which depend on the isolation level. Since the repeatable read isolation level keeps all locks acquired until the end of the transaction, the locks acquired in the first step are held and there is no need to acquire further locks in the second step. For the read stability and cursor stability isolation levels, locks must be acquired during the second step. To maximize concurrency, we don't acquire locks during the first step and rely on the reapplication of all predicates to ensure that only qualifying rows are returned.

Table 7. Lock Modes for Index Scans used for Deferred Data Page Access

Isolation Level	Read-only	Intent to Change	Change
Access Method: *Index Scan with no predicates*			
RR	IS / S	IX / S	X

Table 7. Lock Modes for Index Scans used for Deferred Data Page Access (continued)

Isolation Level	Read-only	Intent to Change	Change
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
Access Method: *Deferred Data Page Access, after an index scan with no predicates*			
RR	IN	IX / S	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
Access Method: *Index Scan with predicates*			
RR	IS / S	IX / S	IX / S
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
Access Method: *Index Scan with start and stop predicates only*			
RR	IS / S	IX / S	IX / X
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
Access Method: *Deferred Data Page Access, after an index scan with predicates*			
RR	IN	IX / S	IX / S
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

The access path is not controlled by the user; it is chosen by the Optimizer.

The access path used can affect the mode and granularity of a lock. For example, in an application using the repeatable read (RR) isolation level, an UPDATE query that uses a table scan without predicates, would use an X lock on the table. If rows were located through an index, the database manager might choose to lock individual rows of the table.

Declared Temporary Tables and Locking

Declared temporary tables are not locked since they are only available to the application that declared them. This type of table only exists from the time that the application declares it until the application completes or disconnects.

LOCK TABLE Statement

You can override the rules for acquiring initial lock modes by using the LOCK TABLE statement in an application.

The statement locks an entire table. Only the table specified in the LOCK TABLE statement is locked. Parent and dependent tables of the specified table are not locked. You must determine whether locking other tables that can be accessed is necessary to achieve the desired result in terms of concurrency and performance. The lock is not released until the unit of work is committed or rolled back.

If a table is normally shared among several users, you might want to lock it for the following reasons:

LOCK TABLE IN SHARE MODE

You want to access data that is *consistent in time*; that is, data current for a table at a specific point in time. If the table experiences frequent activity, the only way to ensure that the entire table remains stable is to lock it. For example, your application wants to take a snapshot of a table. However, during the time your application needs to process some rows of a table, other applications are updating rows you have not yet processed. This is allowed with repeatable read, but this action is not what you want.

As an alternative, your application can issue the LOCK TABLE IN SHARE MODE statement: no rows can be changed, regardless of whether you have retrieved them or not. You can then retrieve as many rows as you need, knowing that the rows you have retrieved have not been changed just before you retrieved them.

With LOCK TABLE IN SHARE MODE, other users can retrieve data from the table, but they cannot update, delete, or insert rows into the table.

LOCK TABLE IN EXCLUSIVE MODE

You want to update a large part of the table. It is less expensive and more efficient to prevent all other users from accessing the table than it is to lock each row as it is updated, and then unlock the row later when all changes are committed.

With LOCK TABLE IN EXCLUSIVE MODE, all other users are locked out; no other applications can access the table unless they are uncommitted read applications.

For more details on the LOCK TABLE statement, refer to the *SQL Reference* manual.

An alternative to the use of the LOCK TABLE statement is the ALTER TABLE statement with the LOCKSIZE parameter. The LOCKSIZE parameter allows for the selection of either ROW locks or TABLE locks. Whatever choice is made becomes the granularity of the locks chosen when the table is next accessed. The selection of ROW locks is no different from selecting the default lock size when a table is created. The selection of TABLE locks may improve the performance of queries by limiting the number of locks that need to be acquired. However, concurrency may be reduced since all locks are held over the complete table. Selecting either choice does not prevent normal lock escalation from occurring. For more details on the ALTER TABLE statement, refer to the *SQL Reference* manual.

CLOSE CURSOR WITH RELEASE

When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, the database manager will attempt to release all read locks (if any) that have been held for the cursor. Read locks are IS, S, and U table locks as well as S, NS, and U row locks. For more information on lock modes, see “Attributes of Locks” on page 49.

The WITH RELEASE clause has no effect for cursors that are operating under the CS or UR isolation levels. When specified for cursors that are operating under the RS or RR isolation levels, the WITH RELEASE clause ends some of the guarantees of those isolation levels. Specifically, a RS cursor may experience the *nonrepeatable read* phenomenon, and a RR cursor may experience either the *nonrepeatable read* or *phantom read* phenomenon.

If a cursor that is originally RR or RS is reopened after being closed using the WITH RELEASE clause, then new read locks will be acquired.

See “DECLARE CURSOR WITH HOLD Statement” on page 75 for a comparison with the other primary clause for the CLOSE CURSOR statement.

The DB2 CLI connection attribute SQL_ATTR_CLOSE_BEHAVIOR can be used in CLI applications to achieve the same results. Refer to the `SQLSetConnectAttr()` section of the *CLI Guide and Reference* for more information.

Summary of Locking Considerations

The following are points to remember about locking:

- Small units of work (frequent COMMIT statements) promote concurrent access of data by many users. Include COMMIT statements when your application is logically at a point of consistency; that is, when the data you have changed is consistent. When a COMMIT is issued, locks are released (except for table locks associated with cursors declared WITH HOLD).

- Locks are acquired even if your application merely reads rows, so it is still important to commit read-only units of work. This is because shared locks are acquired by repeatable read, read stability, and cursor stability isolation levels in read-only applications. With repeatable read and read stability, all locks are held until a COMMIT is issued, preventing other processes from updating the locked data, unless you close your cursor using the WITH RELEASE clause. In addition, catalog locks are acquired even in uncommitted read applications using dynamic SQL.
- The database manager ensures that your application does not retrieve uncommitted data (rows that have been updated by other applications but are not yet committed) unless you are using the uncommitted read isolation level.
- You can lock the entire table that you want to protect by issuing a LOCK TABLE statement:
  - To allow other applications to retrieve, but not update, delete, or insert rows
  - To prevent other applications (other than those with an uncommitted read isolation level) from accessing the rows of a table.
- When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, the database manager will attempt to release all read locks (if any) that have been held for the cursor.
- When changing the configuration parameters affecting locking in a partitioned database, ensure that the changes are made to all of the partitions in the database.

Adjusting the Optimization Class

When an SQL query is compiled, a number of optimization techniques can be used to determine the most efficient access plan for that query. Using more optimization techniques results in:

1. Improvements in run-time performance
2. Increased query compilation time
3. Increased system resource usage.

For this reason, you may wish to limit the number of techniques applied to optimizing your query by setting the optimization class. This can be particularly useful if you have:

- Very small databases or very simple dynamic queries
- Limited memory available at compile time on your database server
- A desire to reduce the query compilation (for example, PREPARE) time.

You may select from any of the query optimization classes described below, although class 0 and class 9 should be used only in special circumstances. Class 5 is the default. Classes 0, 1, and 2 use the Greedy join enumeration algorithm; for complex queries this algorithm considers far fewer alternative plans, and incurs significantly less compilation time, than classes 3 and above. Classes 3 and above use the Dynamic Programming join enumeration algorithm; this algorithm considers far more alternative plans, and can incur significantly more compilation time, than classes 0, 1, and 2 as the number of tables increases.

0 - This class directs the optimizer to use a minimal amount of optimization to generate an access plan. For example:

- Any non-uniform distribution statistics are not considered by the optimizer.
- Only basic query rewrite rules are applied (see "Rewrite Query by the SQL Compiler" on page 143 for information about query rewrite).
- Greedy join enumeration occurs (see "Search Strategies for Selecting Optimal Join" on page 170).
- Only nested loop join and index scan access methods are enabled (see "Join Concepts" on page 165 and "Index Scan Concepts" on page 153).
- List prefetch and index ANDing are disabled so that they are not used in generated access methods.
- The star join strategy is not considered.

This class should only be used in special circumstances requiring the lowest possible query compilation overhead. An application consisting entirely of very simple dynamic SQL statements which access well-indexed tables is a good example of where query optimization class 0 is appropriate.

1 - This class directs the optimizer to use a degree of optimization which is roughly comparable to DB2/6000 Version 1, plus some additional low cost features not found in Version 1. In particular:

- Any non-uniform distribution statistics are not considered by the optimizer.
- Only a subset of the query rewrite rules are applied, including those provided in DB2/6000 Version 1.
- Greedy join enumeration (see "Search Strategies for Selecting Optimal Join" on page 170.)
- List prefetch and index ANDing are disabled so that they are not used in generated access methods.

  Note: Index ANDing is still used when working with the semi-joins found with star-joins.

Optimization class 1 is quite similar to class 0 except that Merge Scan joins and table scans are also available.

2 - This class directs the optimizer to use a degree of optimization which significantly improves upon that of class 1, while keeping the compilation cost significantly lower than classes 3 and above for complex queries. In particular:

- All available statistics, including both frequency and quantile non-uniform distribution statistics, are utilized.
- All of the query rewrite rules are applied, except computationally intensive rules which are applicable only in very rare cases.
- Greedy join enumeration (see "Search Strategies for Selecting Optimal Join" on page 170) is used.
- A wide range of access methods are considered, including list prefetch.
- The star join strategy is considered, if applicable.

Optimization class 2 is quite similar to class 5 except that it uses Greedy join enumeration rather than Dynamic Programming. This class has the most optimization of all the optimization classes that use the Greedy join enumeration algorithm, which considers fewer alternatives for complex queries, and therefore consumes less compilation time than classes 3 and above. It is therefore recommended for very complex queries in a decision support or online analytic processing (OLAP) environment. In such cases, there is a good chance the same query is executed infrequently, so that its access plan is unlikely to remain in the cache until the next occurrence of the query.

3 - This class requests that a moderate amount of optimization be performed to generate an access plan. This class comes closest to matching the query optimization characteristics of DB2 for MVS/ESA or OS/390. This optimization class has the following characteristics:

- Non-uniform distribution statistics, which track frequently occurring values are used, if available.
- Most query rewrite rules, including subquery-to-join transformations are applied.
- Dynamic programming join enumeration (see "Search Strategies for Selecting Optimal Join" on page 170):
  – Limited use of composite inner tables (see "Composite Tables" on page 172)
  – Limited use of Cartesian products for star schemas involving "look-up" tables (see "Search Strategies for Star Join" on page 170)
- A wide range of access methods are considered, including list prefetch, index ANDing and star joins.

This class is suitable for a broad range of applications. Using this class gives the optimizer a better chance of selecting an excellent access plan for queries with four or more joins. However, the optimizer might fail to consider a better plan which would be chosen with the default query optimization class.

5 - This class directs the optimizer to use a significant amount of optimization to generate an access plan. For example, class 5 has the following characteristics:

- All available statistics including both frequency and quantile non-uniform distribution statistics.
- All of the query rewrite rules are applied including the routing of queries to summary tables, except for those computationally intensive rules which are applicable only in very rare cases.
- Dynamic programming join enumeration (see "Search Strategies for Selecting Optimal Join" on page 170):
  - Limited use of composite inner tables (see "Composite Tables" on page 172)
  - Limited use of Cartesian products for star schemas involving "look-up" tables (see "Search Strategies for Star Join" on page 170)
- A wide range of access methods are considered, including list prefetch, index ANDing, and summary table routing.

When the optimizer detects that the additional resources and processing time are not warranted for complex dynamic SQL queries, optimization is reduced. The extent or size of the reduction is dependent on the machine size and the number of predicates.

When the query optimizer reduces the amount of query optimization performed, it continues to apply all the query rewrite rules that would normally be applied. However, it does use the greedy join enumeration method and reduces the number of access plan combinations that are considered.

Query optimization class 5 is an excellent choice for a mixed environment consisting of both transactions and complex queries. This optimization class has been designed to apply the most valuable query transformations and other query optimization techniques in an efficient manner.

7 - This class directs the optimizer to use a significant amount of optimization to generate an access plan. It is the same as query optimization class 5 except that it does not reduce the amount of query optimization for complex dynamic SQL queries.

9 - This class directs the optimizer to use all available optimization techniques. These include:
- All available statistics
- All query rewrite rules
- All possibilities for join enumerations, including Cartesian products and unlimited composite inners
- All access methods.

This class can greatly expand the number of possible access plans that are considered by the optimizer. This class should be used to determine whether more comprehensive optimization can generate a better access plan for very complex and very long-running queries using large tables. Explain and performance measurements should be used to verify that a better plan has been found.

How Do You Set the Optimization Class?

The way to request a specific query optimization class depends on whether you are using static or dynamic SQL.

- *Static SQL statements* use the optimization class specified on the PREP and BIND commands. The QUERYOPT column in the SYSCAT.PACKAGES catalog table records the optimization class used to bind the package. If the package is rebound either implicitly or using the REBIND PACKAGE command, this same optimization class will be used for the static SQL statements. If you want to change the optimization class used for these static SQL statements, you must use the BIND command. If you do not specify the optimization class, DB2 uses the default optimization as specified by *dft_queryopt*.
- *Dynamic SQL statements* use the optimization class specified by the CURRENT QUERY OPTIMIZATION special register which is set using the SQL SET statement. For example, the following statement sets the optimization class to 1:

  ```
  SET CURRENT QUERY OPTIMIZATION = 1
  ```

  To ensure that a dynamic SQL statement always uses the same optimization class, you may want to include this SET statement in your application program. For more information, refer to the *SQL Reference*.

  If the CURRENT QUERY OPTIMIZATION register has not been set, dynamic statements will be bound using the default query optimization class. The default value for both dynamic and static SQL is determined by value of the configurable database parameter DFT_QUERYOPT. Class 5 is the default query optimization class unless you have changed the default. (For more information on this parameter, see “Default Query Optimization

Class (dft_queryopt)" on page 421.) The default values for the bind option and the special register are taken from the DFT_QUERYOPT configuration parameter.

How Much Optimization is Necessary?

Most statements will be adequately optimized using a reasonable amount of resources with the default query optimization class. The query compilation time and resource consumption, at a given optimization class, is primarily influenced by the complexity of the query, particularly the number of joins and subqueries. However, compilation time and resource usage are also affected by the amount of optimization performed for the various optimization classes. For any optimization class, you can expect to see a greater difference in query compilation time and resource consumption for a very complex query than for a simple one.

The following may help you select which optimization class to use:

- Start by using the default query optimization class.
- If you wish to use a class other than the default, try class 1, 2 or 3 first.
- Use a low optimization class (0 or 1) for queries having very short run-times, that is, queries taking less than one second. (See the following discussion for additional criteria about when to choose a low optimization class.)
- Use optimization class 1 or 2 if you have many tables with many of the join predicates that are on the same column, and if compilation time is a concern.
- Use a higher optimization class (3, 5, or 7) for long running queries, that is, queries taking more than 30 seconds.
- Under normal circumstances, you should not use optimization class 9.
- For queries that run a long time, run the query using db2batch to determine how much of the time is spent in compilation and how much is spent in execution.
  - If most of the time is spent in compilation then reduce the optimization class.
  - If most of the time is spent in execution then consider a higher optimization class.

Note that query optimization classes 1, 2, 3, 5, and 7 are all suitable for general purpose use.

Only if you require further reductions in query compilation time and you know the kind of SQL (for example, extremely simple statements) that will be executed should you consider class 0. This SQL will tend to have the following characteristics:

- Access to a single or only a few tables

- Fetches a single or only a few rows
- Uses fully qualified, unique indexes.

Online transaction processing (OLTP) transactions are good examples of this kind of SQL.

Complex queries may require different amounts of optimization to select the best access plan. You may wish to consider using higher optimization classes for queries exhibiting the following characteristics:
- Access to large tables
- A large number of predicates
- Many subqueries
- Many joins
- Many set operators, such as UNION and INTERSECT
- Many qualifying rows
- GROUP BY and HAVING operations
- Nested table expressions
- A large number of views.

Decision support queries or month-end reporting queries against fully normalized databases are good examples of complex queries where at least the default query optimization class should be used.

Another reason to use higher query optimization classes is SQL which was produced by a query generator. Many query generators create SQL which is not efficient. Poorly written queries, including those produced by a query generator, may require additional optimization to make it possible to select a good access plan. Using query optimization class 2 and higher can improve poorly written SQL queries.

The use of static or dynamic SQL, and whether the same dynamic SQL is repeatedly executed are also important considerations. For static SQL, the query compilation time and resources are expended just once and the resulting plan can be used many times. In general, static SQL should always use the default query optimization class. Dynamic statements are bound and executed at run time; therefore, you should consider whether the overhead of additional optimization for dynamic statements improves your overall performance. However, if the same dynamic SQL statement is executed repeatedly, the selected access plan will be cached. For the purposes of selecting a query optimization class, the statement can be treated like a static SQL statement.

(Refer to the *Application Development Guide* for information on when to use static and dynamic SQL.)

If you think you have a query that could benefit from additional optimization, but you are not sure, or you are concerned about compilation time and resource usage, you may want to perform some benchmark testing. This testing can help you quantify the benefits obtained from different optimization classes. See “Chapter 12. Benchmark Testing” on page 299 for general techniques and the specific use of the `db2batch` tool. When designing and running your benchmark test, consider whether the SQL statements in your application are static or dynamic:

- For **dynamic** SQL statements, your testing should compare the average run time for the statement. You can use the following formula to help you calculate the average run time:

  ```
  compile time + sum of execution times for all iterations
  --------------------------------------------------------
                number of iterations
  ```

  where, the `number of iterations` represents the number of times that you expect that the SQL statement will be executed each time it is compiled.

  Note: Following the initial compilation, dynamic SQL statements are recompiled when a change to the environment requires the statement to be recompiled. Once cached, a SQL statement does not need to be compiled again since subsequent PREPARE statements will re-use the cached statement assuming the environment does not change. (See “Catalog Cache Size (catalogcache_sz)” on page 331 and “Package Cache Size (pckcachesz)” on page 338 for information about a cache that can improve performance when working with dynamic SQL statements.)
- For **static** SQL statements, your testing should compare the statement run times.

  Note: While you may also be interested in the compile time of static SQL, the total (compile and run) time for the statement is difficult to use in any meaningful context. Comparing the total time does not recognize the fact that a static SQL statement can be run many times for each time it is bound and that it is generally not bound during run time.

Restrictions on Result Sets to Improve Performance

A SELECT statement defines a set of rows which satisfy the search criteria. The DB2 optimizer assumes the application will retrieve all the qualifying rows. This assumption is most appropriate in OLTP and batch environments. However, in “browse” applications it is common for a query to define a very large potential answer set but only retrieve the first few rows, typically only as many rows as are required to fill the screen.

The default assumption made by the optimizer to retrieve all qualifying rows may not be the best for applications that are not updating or deleting information from the stored data.

There are four ways of modifying the SELECT statement to limit or modify the result table to improve performance. They are:

- FOR UPDATE clause
- FOR READ/FETCH ONLY clause
- OPTIMIZE FOR n ROWS clause
- FETCH FIRST n ROWS ONLY clause.

FOR UPDATE Clause

The FOR UPDATE clause identifies the columns that can be updated by a subsequent positioned UPDATE statement. If the FOR UPDATE clause is specified without column names, all columns that can be updated in the table or view are included. If column names are specified, each name must be unqualified and must identify a column of the table or view.

The FOR UPDATE clause cannot be used when either of the following are true:

- The cursor associated with the SELECT statement cannot be deleted.
- At least one of the selected columns is a column that can not be updated in a catalog table and has not been excluded in the FOR UPDATE clause.

The DB2 CLI connection attribute SQL_ATTR_ACCESS_MODE can be used in CLI applications to achieve the same results. Refer to the `SQLSetConnectAttr()` section of the *CLI Guide and Reference* for more information.

FOR READ or FETCH ONLY Clause

The FOR READ ONLY clause ensures that the result table is read-only. The FOR FETCH ONLY clause has the same meaning.

Some result tables are read-only by definition. For example, the result table from a SELECT on a view defined as read-only. You can still specify FOR READ ONLY in such a case, but the clause has no effect.

For result tables where updates and deletes are allowed, specifying FOR READ ONLY may improve the performance of FETCH operations. This possible improvement in performance occurs when the database manager is able to do blocking, rather than exclusive locks, on the data. You should use the FOR READ ONLY clause to improve performance except in cases where queries are used in positioned UPDATE or DELETE statements.

The DB2 CLI connection attribute SQL_ATTR_ACCESS_MODE can be used in CLI applications to achieve the same results. Refer to the `SQLSetConnectAttr()` section of the *CLI Guide and Reference* for more information.

OPTIMIZE FOR n ROWS Clause

The OPTIMIZE FOR clause provides a mechanism for an application to declare its intent to retrieve only a subset of the result or to give priority to the retrieval of the first few rows. Once this intent is understood, the optimizer can give preference to access plans that minimize the response time for retrieving the first few rows. Also, the number of rows that are sent to the client as a single block (see “Row Blocking” on page 76) are bounded by the value of “n” in the OPTIMIZE FOR clause. Therefore, the OPTIMIZE FOR clause affects both how the qualifying rows are retrieved from the database by the server, and how the qualifying rows are returned to the client.

For example, suppose you are querying the employee table for the employees with the highest salary on a regular basis.

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
FROM EMPLOYEE
ORDER BY SALARY DESC
```

You have defined a descending index on the SALARY column. However, since employees are ordered by employee number, the salary index is likely to be very poorly clustered. The optimizer, in trying to avoid many random synchronous I/Os, would likely choose to use the list prefetch access method (see “Understanding List Prefetching” on page 243) which requires the row identifiers of all rows that qualify to be sorted. This can cause a delay before the first qualifying rows can be returned to the application. By adding the OPTIMIZE FOR clause to the statement as follows:

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
FROM EMPLOYEE
ORDER BY SALARY DESC
OPTIMIZE FOR 20 ROWS
```

the optimizer would likely choose to use the SALARY index directly with the knowledge that in all likelihood only the twenty employees with the highest salaries would be retrieved. Regardless of how many rows could be blocked, a block of rows is returned to the client every twenty rows.

Use of the OPTIMIZE FOR clause causes the optimizer to favor access plans that avoid bulk operations or operations that interrupt the flow of rows, such as sorts. You are most likely to influence an access path by using OPTIMIZE FOR 1 ROW. As a result, using this clause could have the following effects:

- Join sequences with composite inners are less likely since they require a temporary table.

- The join method could change. A nested loop join is the most likely choice, because it has low overhead cost and is usually more efficient if you only want to retrieve a few rows.
- An index that matches the ORDER BY clause is more likely to be picked. This occurs because no sort would be needed for the ORDER BY.
- List prefetch is less likely to be picked since this access method requires a sort.
- Sequential prefetch is less likely to be requested by DB2 because it infers that you only want to see a small number of rows.
- In a join query, the table with the columns in the ORDER BY clause is likely to be picked as the outer table if there is an index on that outer table that gives the ordering needed for the ORDER BY clause.

Although the OPTIMIZE FOR clause applies to all optimization classes (see "Adjusting the Optimization Class" on page 64), it works best for optimization class 3 and higher. The use of the greedy join enumeration method (see "Search Strategies for Selecting Optimal Join" on page 170) in optimization classes below 3 sometimes results in access plans for multi-table joins that do not lend themselves to quickly retrieving the first few rows.

The OPTIMIZE FOR clause does not prevent you from retrieving all the qualifying rows. However the total elapsed time to retrieve all the qualifying rows may be significantly greater than if the optimizer had been allowed to optimize for the entire answer set.

If you have a packaged application that uses the call level interface (DB2 CLI or ODBC) it is possible to have DB2 CLI automatically append an OPTIMIZE FOR clause to the end of each query statement using the OPTIMIZEFORNROWS keyword in the `db2cli.ini` configuration file. For additional information refer to the *CLI Guide and Reference* manual.

When selecting data from nicknames, results may vary depending on data source support. If the data source referenced by the nickname supports the OPTIMIZE FOR clause, and the DB2 optimizer pushes down the entire query containing the clause to the data source, then the clause is generated in the remote SQL sent to the data source. If the data source does not support this clause, or if the optimizer decides to execute the clause locally (least cost plan), the OPTIMIZE FOR clause is applied locally at DB2. In this case, the DB2 optimizer will continue to give preference to access plans that minimize the response time for retrieving the first few rows of a query, but the options available to the optimizer for generating plans are slightly delimited and performance gains from the OPTIMIZE FOR clause may be negligible.

If both the FETCH FIRST clause and the OPTIMIZE FOR clause are specified, the lower of the two values is used to influence the communications buffer size. The two values are considered independent of each other for optimization purposes. See “Using a SELECT-Statement” on page 77 for more information on the interaction between these two clauses.

FETCH FIRST n ROWS ONLY Clause

The OPTIMIZE FOR *n* ROWS clause does not prevent the retrieval of all qualifying rows. (The total elapsed time to retrieve all qualifying rows may be significantly greater than if the optimizer was allowed to optimize for the entire answer set.)

The FETCH FIRST *n* ROWS ONLY clause sets the maximum number of rows that can be retrieved from within a SELECT statement. Limiting the result table to the first several rows can improve performance. Only n rows are retrieved regardless of the number of rows there might be in the result table based on a SELECT where this clause is not specified.

If both the FETCH FIRST clause and the OPTIMIZE FOR clause are specified, the lower of the two values is used to influence the communications buffer size. The two values are considered independent of each other for optimization purposes. See “Using a SELECT-Statement” on page 77 for more information on the interaction between these two clauses.

DECLARE CURSOR WITH HOLD Statement

When you declare a cursor with the DECLARE CURSOR statement that includes the WITH HOLD clause, any open cursors remain open when the transaction is committed. Further, all locks are released, except locks protecting the current cursor position of open WITH HOLD cursors.

When you declare a cursor with the DECLARE CURSOR statement that includes the WITH HOLD clause, all open cursors are closed when the transaction ends with a ROLLBACK. Further, all locks are released and LOB locators are freed.

See “CLOSE CURSOR WITH RELEASE” on page 63 for a comparison with the other primary clause for the CLOSE CURSOR statement.

The DB2 CLI connection attribute SQL_ATTR_CURSOR_HOLD can be used in CLI applications to achieve the same results. For additional information refer to the “SQLSetStmtAttr - Set Options Related to a Statement” section in the *CLI Guide and Reference* manual.

If you have a packaged application that uses the call level interface (DB2 CLI or ODBC) it is possible to have DB2 CLI automatically assume the WITH HOLD clause for every declared cursor by using the CURSORHOLD keyword in the `db2cli.ini` configuration file. Refer to the transaction configuration keywords section of the *CLI Guide and Reference* for more information.

Row Blocking

Row blocking is a technique that reduces database manager overhead by retrieving a *block* of rows in a single operation. These rows are stored in a cache, and each FETCH request in the application gets the next row from the cache. When all the rows in a block have been processed, another block of rows is retrieved by the database manager.

The cache is allocated when an application issues an OPEN CURSOR request and is deallocated when the cursor is closed. The size of the cache is determined by a configuration parameter which is used to allocate memory for the I/O block. The parameter used depends on whether the client is local or remote:

- For *local applications*, the parameter *aslheapsz* is used to allocate the cache for row blocking. (See "Application Support Layer Heap Size (aslheapsz)" on page 353 for information about this parameter.)
- For *remote applications*, the parameter *rqrioblk* on the client workstation is used to allocate the *cache* for row blocking. The cache is allocated on the database client. (See "Client I/O Block Size (rqrioblk)" on page 355 for information about this parameter.)

For *local* applications, you can use the following formula to estimate how many rows are returned per block, where:
- *aslheapsz* is in pages of memory
- 4 096 is the number of bytes per page
- *orl* is the output row length in bytes:

```
Rows per block = aslheapsz * 4096 / orl
```

For *remote* applications, you can use the following formula to estimate how many rows are returned per block, where:
- *rqrioblk* is in bytes of memory
- *orl* is the output row length in bytes:

```
Rows per block = rqrioblk / orl
```

Note that if you use the FETCH FIRST *n* ROWS ONLY clause or the OPTIMIZE FOR *n* ROWS clause in a SELECT statement, the number of rows per block will be the minimum of the following:
- The value calculated in the above formula
- The value of *n* in the FETCH FIRST clause
- The value of *n* in the OPTIMIZE FOR clause

Use the BLOCKING option on the PREP and BIND commands to specify one of the following types of row blocking:

UNAMBIG	Blocking occurs for read-only cursors and cursors not specified as "FOR UPDATE OF". Ambiguous cursors are treated as updateable.
ALL	Blocking occurs for read-only cursors and cursors not specified as "FOR UPDATE OF". Ambiguous cursors are treated as read-only.
NO	Blocking does not occur for any cursors. Ambiguous cursors are treated as read-only.

For details of these types of row blocking, refer to the PREP and BIND command descriptions in the *Command Reference* manual.

If no option is specified on the PREP and BIND commands, the default row blocking type is UNAMBIG. For the command line processor and call level interface, the default row blocking type is ALL.

Refer to the *SQL Reference* for more information about cursors.

Tuning Queries

This section provides specific considerations and guidelines to help you fine-tune the SQL statements in an application program. As a general rule, these guidelines may help design a program that minimizes the use of system resources and the amount of time needed to access data in a very large table. Depending on the amount of optimization that takes place when the SQL statement is compiled, you may not need to fine-tune your SQL statements. The SQL compiler can rewrite your SQL into more efficient forms. See "Rewrite Query by the SQL Compiler" on page 143 and "Adjusting the Optimization Class" on page 64.

It is also important to note that the access plan chosen by the optimizer is also affected by other factors, including environmental considerations and system catalog statistics. If you conduct benchmark testing of the performance of your applications, you can make adjustments that can improve the access plan.

Using a SELECT-Statement

The SQL language is a high-level language with much flexibility. As a result, different *select-statements* can be written to retrieve the same data. However, the performance can vary for the different forms and the different classes of optimization.

The SQL compiler (including the query rewrite and optimization phases) will choose an access plan to produce the result set for the query you have coded. Therefore, as noted in many of the following guidelines, you should code your query to obtain only the data that you need.

Guidelines When Using a SELECT-Statement

The guidelines for using a *select-statement* are:

- Specify only those columns that are needed in the select list. Although it may be simpler to specify all columns with an asterisk (*), needless processing and returning of unwanted columns can result.
- Limit the number of rows selected by using predicates to restrict the answer set to only those rows that you require. (See “Predicate Terminology” on page 162 for more information about the different types of predicates and their relative impact on performance.)
- When the number of rows you want to use is significantly less than the total number of rows that could be returned, specify the OPTIMIZE FOR clause for the *select-statement*. This clause affects both the choice of access plans as well as the number of rows that are blocked in the communication buffer. (For more information, see “Row Blocking” on page 76.)
- When the number of rows to be retrieved is small, there is no need to specify the OPTIMIZE FOR k ROWS clause in addition to the FETCH FIRST n ROWS ONLY clause. However, if n is large and you want optimize by getting the first k rows quickly with a possible delay for the subsequent k rows, specify both. The communication buffers are sized based on the lesser of n and k.

  ```
  SELECT EMPNAME, SALARY FROM EMPLOYEE
     ORDER BY SALARY DESC
     FETCH FIRST 100 ROWS ONLY
     OPTIMIZE FOR 20 ROWS
  ```
- Specifying the FOR READ ONLY (or FOR FETCH ONLY) clause can improve performance by allowing your query to take advantage of row blocking. It can also improve data concurrency since exclusive locks will never be held on the rows retrieved by a query with this clause specified. It also allows additional query rewrites to take place. Specifying the FOR READ ONLY (or FOR FETCH ONLY) clause along with BLOCKING ALL BIND can similarly improve the performance of queries against nicknames in a federated system.
- Specifying the FOR UPDATE OF clause can also improve performance, for cursors that will be updated, by allowing the database manager to initially choose more appropriate locking levels, thus avoiding potential deadlocks (see “Deadlocks” on page 56) and lock conversions (see “Lock Conversion” on page 53).
- Avoid numeric data type conversions whenever possible. When comparing values, it may be more efficient to use items that have the same data type. If conversions are necessary, inaccuracies due to limited precision, and performance costs due to run-time conversions, may result.

  If possible, use the following data types:
  - Character rather than varying character for short columns
  - Integer rather than float or decimal

  - Datetime rather than character.
  - Numeric rather than character.
- SQL statements containing clauses or operations such as DISTINCT, or ORDER BY, require data to be ordered to perform the operation. If you want to decrease the chances that a sort operation will be used, omit the specification of these clauses if they are not required.
- To check for existence of rows in a table, do not use:
  ```
  SELECT COUNT(*) FROM  TABLENAME
  ```
  and check for a value of nonzero unless you know that the table will be very small. As the table gets larger, counting all the rows will impact performance. Instead it is suggested that you try to select a single row. This can be done by either opening a cursor and fetching one row, or by doing a single-row (SELECT INTO) selection. (Remember to check for the SQLCODE -811 error if more than one row is found from the *select-statement*.)
- If update activity is low and your tables are large, define indexes on columns that are frequently used as predicates.
- Consider using an IN list if the same column appears in multiple predicate clauses.
- For large IN lists used in conjuction with host variables, loopin an a subset of the host variables may improve performance.

The following suggestions apply specifically to *select-statements* that access several tables.
- Use join predicates when joining tables. (A join predicate is a comparison between two columns from different tables in a join.)
- Define indexes on the columns in the join predicate to allow the join to be processed more efficiently. This will also benefit UPDATE and DELETE statements that contain select-statements that access several tables.
- If possible, avoid using expressions or OR clauses with join predicates. In this case, some join techniques cannot be used by the database manager and, as a result, the most efficient join method may not be chosen.
- If possible, ensure that the tables joined are both partitioned on the join column in a partitioned database environment.

For more information see “Join Concepts” on page 165.

Also, refer to the *Application Development Guide* for more information on coding SQL statements with joins and subqueries.

Compound SQL

Compound SQL allows you to group several SQL statements into a single executable block. The SQL statements contained within the block (*sub-statements*) could be executed individually; however, by creating and executing a block of statements, you reduce the database manager overhead. For remote clients, compound SQL also reduces the number of requests that have to be transmitted across the network.

There are two types of compound SQL:

- **Atomic**

  The application receives a response from the database manager when all sub-statements have completed successfully, or when one sub-statement ends in an error. If one sub-statement ends in an error, the entire block is considered to have ended in an error, and any changes made to the database within the block will be rolled back.
- **Not Atomic**

  The application receives a response from the database manager when all sub-statements have completed. All sub-statements within a block are executed regardless of whether or not the preceding sub-statement completed successfully. The group of statements can only be rolled back if the unit of work containing the NOT ATOMIC compound SQL is rolled back.
- Atomic compound SQL is not supported with DB2 Connect
- Compound SQL is supported within stored procedures (also known as DARI routines)
- Compound SQL is supported through:
  - Embedded static SQL (refer to the *SQL Reference* manual)
  - DB2 Call Level Interface (refer to the *CLI Guide and Reference* manual)
  - JDBC (refer to the *Application Development Guide* manual.

Performance Considerations and Character Conversion

When your application and database are not using the same code page, a mapping of the data from one code page to the other code page takes place, if possible. To properly map data between application and database code pages, some data conversion may be required.

This mapping and data conversion introduce a certain amount of overhead into the processing time for applications that are running in a code page that is different from the database code page. Your application's performance can be improved if the application and database are using the same code page or the identity collating sequence.

Code Page Conversion

Character conversion can occur in the following situations:

- When a client or application accessing a database is running in a code page that is different from the code page of the database.
  - Database conversion will occur on the database server machine: From the application code page to the database code page; and, from the database code page to the application code page.
- When a client or application importing (or loading) a file runs in a code page different from the file being imported (or loaded).
- When DB2 Connect is used to access data on a DRDA server.

Character conversion will **not** occur for:

- File names.
- Data targeted for, or coming from, a column assigned the FOR BIT DATA attribute, or data used in an SQL operation whose result is FOR BIT or BLOB data.
- A DB2 product or platform that does not have a supported conversion function to, or from, EUC or UCS-2 installed. You receive an SQLCODE -332 (SQLSTATE 57017) when running your application.

For more information about EUC code page support and National Language Support (NLS) considerations, refer to the *Administration Guide: Planning*.

Depending on the operating system environment DB2 database managers use a conversion function and conversion tables, or DBCS conversion APIs, when converting multi-byte code pages.

Note: Character string conversions between multi-byte code pages, like DBCS with EUC, may result in either an increase or a decrease in the length of the string.

Code points assigned to different characters in a country's PC DBCS, EUC, and UCS-2 code sets may produce different results when sorting the same characters. If sorting is required across code sets for different countries, you should refer to the *Administration Guide: Planning*.

Extended UNIX Code (EUC) Code Page Support

Use of host variables that use graphic data in C or C++ applications require special considerations including special precompiler, application performance, and application design issues.

If applications are developed requiring EUC code sets, you should see the *Administrative API Reference* manual.

Database and client application support for graphic (that is, double byte character) data must overcome the two bytes wide restriction when dealing with many characters found in both the Japanese and Traditional Chinese EUC code pages. Graphic data from these EUC code pages is stored and manipulated using the UCS-2 code set.

Stored Procedures

In a database application environment, many situations are repetitive; for example, receiving a fixed set of data, performing the same multiple requests against a database, or returning a fixed set of data. Stored procedures permit one call to a remote database to execute a preprogrammed procedure. One call may represent several accesses to the database.

Processing a single SQL statement for a remote database requires sending two transmissions: one request and one receive. However, an application can contain many SQL statements. Without stored procedures, many transmissions are required for an application to complete its work.

When a database client uses a stored procedure, it requires only two transmissions for the entire process, thereby reducing the number of network transmissions. To invoke a stored procedure, the requesting application must connect to the database containing the procedure before calling it.

Typically these stored procedures are run in processes separate from the database agents. This separation requires that the stored procedure and agent processes must communicate through a router. To obtain the best possible performance for a stored procedure, it is possible to identify a stored procedure as being “trusted”, or “not fenced”, and as a result, run the procedure directly in the database agent process. What do we mean by “trusted” and “not fenced”?

- *Not fenced* refers to the fact that there is nothing separating the stored procedure from the database control structures that are used by the database agent.
- *Trusted* indicates that as an administrator, you are confident that the stored procedure will not accidentally or maliciously damage the database control structures. That is, you trust them to operate in a fashion which will not jeopardize your database integrity.

Both of these terms mean the same thing, that is, if your stored procedure is “not fenced”, then your stored procedure is “trusted”. Due to the associated risk of damaging your database, you **SHOULD ONLY** use not fenced stored procedures when you need to obtain the maximum possible performance benefits. In addition, you should ensure that the procedure is well coded and has been thoroughly tested before allowing it to run as a not fenced stored procedure. If a fatal error does occur while running one of these not fenced stored procedures, the database manager will determine whether the error occurred in the application or database manager code, and perform the appropriate recovery.

It is possible for a not fenced stored procedure to corrupt the database manager beyond recovery, possibly resulting in lost data and/or a corrupt database. Extreme caution should be exercised when running trusted stored procedures. In almost all cases, the proper performance analysis of an application will result in the desired performance without using this option. For example, performance may be improved through the use of triggers.

There are two ways to create a stored procedure as being not fenced:
- Use the CREATE PROCEDURE command and specify the NOT FENCED clause.
- Put the procedure in a special directory, as defined in the *Quick Beginnings* manual for your platform. (This method does not work for Java stored procedures.)

To run a stored procedure, the end-user running the application that calls the procedure must have one of the following privileges at run time:
- EXECUTE or CONTROL privilege for the package associated with the stored procedure
- SYSADM or DBADM authority

For information on writing programs using stored procedures, refer to the *Application Development Guide* manual.

Activating a Database

When a database is started, several types of data are cached. For example, data buffers are cached in the buffer pool, and packages and dynamic SQL statements are cached in the package cache.

If frequent, short periods occur during which no user is connected to the database, and these periods are interspersed with other periods during which a few users are connected to the database, the benefits provided by caching are lost because the cache is frequently destroyed. To avoid this situation, consider activating the database by issuing the following command:

```
DB2 ACTIVATE DATABASE database
```

This command activates the specified database and starts up all necessary services, so that the database is available for connection and use by any application. Databases initialized by ACTIVATE DATABASE can be shut down by DEACTIVATE DATABASE or by *db2stop*. For more information about these commands, refer to the *Command Reference* manual.

Parallel Processing of Applications

A type of parallel environment supported by DB2 is one which requires symmetric multi-processor (SMP) machines. In this environment, more than one processor shares access to the database. This allows parallel execution of complex SQL requests which can be divided among the processors.

You can specify the degree of parallelism to implement when compiling your application by using the CURRENT DEGREE special register, or the DEGREE bind option. ″Degree″ simply refers to the number of concurrently executing parts of a query. There is no strict relation between the number of processors and the value selected for the degree of parallelism. The total number of processors available for use in your hardware platform need not be requested while running your applications; you can select more or less than this number.

Each degree of parallelism adds to the system memory and CPU overhead.

When exploiting parallelism, you should be aware that some configuration parameters require modification in order to optimize performance. Configuration parameters controlling the amount of shared memory and prefetching should be reviewed and modified as necessary in an environment with a high degree of parallelism. See “Parallel” on page 437 for a list of parameters related to parallel operations and partitioned database environments.

There are 3 configuration parameters that you can use to control and manage parallelism. The first, the *intra_parallel* database manager configuration parameter, enables or disables instance parallelism support. The second, the *max_querydegree* database configuration parameter, sets an upper limit for the degree of parallelism for any query in the database. This value overides the CURRENT DEGREE special register and the DEGREE bind option. The third configuration paremeter is the *dft_degree* database configuration parameter. It sets the default value for the CURRENT DEGREE special register and the DEGREE bind option.

For more information on the application use and implications from using more than one degree of parallelism, refer to the *Application Development Guide* manual.

If a query is run with DEGREE = ANY, the database manager chooses the degree of intra-partition parallelism based on a number of factors including the number of processors and the characteristics of the query. The actual degree used at runtime may be lower than the number of processors depending on these factors.

The degree of parallelism is determined by the SQL optimizer when the statement is compiled and may be adjusted before query execution depending on the database activity. The degree of parallelism may be lower than that chosen by the SQL optimizer if the system is heavily utilized. This occurs since intra-partition parallelism aggressively uses system resources to reduce the elapsed time of the query which may adversely affect the performance of other database users.

The degree of parallelism chosen by the SQL optimizer can be found by using the SQL Explain Facility to display the access plan. The degree of parallelism used at runtime can be found by using the database System Monitor. See "Chapter 7. SQL Explain Facility" on page 201 and "Appendix C. SQL Explain Tools" on page 531 for more information on the SQL Explain Facility and related tools. Refer to the *System Monitor Guide and Reference* for additional monitor information.

Note: The ″degree″ of parallelism can be set independent of the hardware environment. This means that you can use a degree of parallelism without having an SMP machine. For example, ″I/O-bound″ queries on a uni-processor machine may benefit from declaring a degree of ″2″ or more. In this case, the uni-processor may not have to wait for input or output tasks to complete before working on a new query. Declaring a degree of ″2″ or more does not directly control I/O parallelism on a uni-processor machine. Utilities such as LOAD can control I/O parallelism independent from such a declaration. The keyword `ANY` can also be used when changing the *dft_degree*. The use of `ANY` means that the optimizer determines the degree of intra-partition parallelism.

In many cases, *database agents* are used to coordinate parallel execution. See "Database Agents" on page 257 for more information, and a list of the various database manager configuration parameters that affect database agents.

Chapter 4. Environmental Considerations

In addition to the factors you should consider when you are designing and coding your application (described in “Chapter 3. Application Considerations” on page 41), there are environmental factors that can influence the access plan chosen for your application:

- Configuration Parameters Affecting Query Optimization
- Nodegroup Impact on Query Optimization
- Table Space Impact on Query Optimization
- Indexing Impact on Query Optimization
- Server Options Affecting Federated Database Queries.

Also see “Chapter 5. System Catalog Statistics” on page 107 for more information about factors that affect the SQL optimizer.

When tuning your applications and environment, you **should** rebind your applications after you make changes in any of the above areas. This ensures that the best access plan is being used.

Configuration Parameters Affecting Query Optimization

Several configuration parameters affect the access plan chosen by the SQL compiler. Many of these are appropriate to a single-partition database and some are only appropriate to a partitioned database. When working with configuration parameters in a partitioned database, it is recommended that the values used for each parameter be the same on all partitions.

When working in a federated system, if the majority of your queries access nicknames then consider the type of query you are sending before changing your environment. For example, the buffer pool does not cache pages from data sources; as such, increasing the *buffpage* parameter value does not guarantee that the optimizer will consider additional alternatives when creating an access plan for queries containing nicknames. (`Data sources` are DBMSs and data within the federated system.) Also, the optimizer may decide that local materialization of data source tables is the least cost route or a necessary step for a sort operation. In that case, increasing the resources available to DB2 Universal Database may speed performance. For additional information, see “Server Options Affecting Federated Database Queries” on page 101 and “Database Shared Memory” on page 326.

Following is a list of configuration parameters that affect the access plan chosen by the SQL compiler:

- "Buffer Pool Size (buffpage)" on page 327.

  When selecting the access plan, the optimizer considers the I/O cost of fetching pages from disk to the buffer pool. In its calculations, the optimizer will estimate the number of I/Os required to satisfy a query. This estimate includes a prediction of buffer pool usage, since additional physical I/Os are not required to read rows in a page that is already in the buffer pool. The optimizer considers the value of the *npages* column in the BUFFERPOOLS system catalog tables in estimating whether a page will be found in the buffer pool.

  The I/O costs of reading the tables can have an impact on :
  – How two tables are joined, as described in "Outer versus Inner Determination" on page 168.
  – Whether an unclustered index will be used to read the data (see "Clustered Indexes" on page 160).

  You can have more than one buffer pool in a database. You can also have more than one buffer pool in a partitioned database. The new buffer pool can be selectively added to each of the partitions in the database or across all partitions. The *npages* column in the BUFFERPOOLS and BUFFERPOOLSNODE system catalog tables are used for estimating in a partitioned database.
- "Default Degree (dft_degree)" on page 421.

  The *dft_degree* configuration parameter specifies the default value for the CURRENT DEGREE special register and the DEGREE bind option. A value of one (1) means no intra-partition parallelism. A value of minus one (-1) means the optimizer determines the degree of intra-partition parallelism based on the number of processors and the type of query.
- "Default Query Optimization Class (dft_queryopt)" on page 421.

  When compiling SQL queries, you can use the query optimization class to direct the optimizer to use different degrees of optimization. For more information on selecting a suitable query optimization class, see "Adjusting the Optimization Class" on page 64.
- "Average Number of Active Applications (avg_appls)" on page 375.

  The *avg_appls* parameter is used by the SQL optimizer to help estimate how much of the buffer pool will be available at run-time for the access plan chosen. Higher values for this parameter can influence the optimizer to choose an access plan for queries that will be more conservative in its buffer pool usage. A value of 1 for this parameter will cause the optimizer to treat the entire buffer pool as being available to the application.
- "Sort Heap Size (sortheap)" on page 342.

  A sort is considered to be "piped" if it does not require a temporary table to store the final, sorted list of data. That is, the results of the sort can be read in a single, sequential access. Piped sorts result in better performance than non-piped sorts and will be used if possible. (See "Influence of Sorting on the Optimizer" on page 180 for a definition of non-piped sorts compared to piped sorts.)

When choosing an access plan, the optimizer estimates the cost of the sort operations, including evaluating whether a sort can be piped, by:
– Estimating the amount of data to be sorted
– Looking at the *sortheap* parameter to determine if there is enough space for the sort to be piped.

- "Maximum Storage for Lock List (locklist)" on page 335 and "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 364.

  When the isolation level (see "Concurrency" on page 41) being used is **repeatable read (RR)**, the SQL optimizer will consider the values of the *locklist* and *maxlocks* parameters to determine whether it is likely that row level locks will be escalated to a table level lock. If the optimizer predicts that lock escalation will occur for a table access, then it will choose a table level lock for the access plan, rather than incurring the overhead of lock escalation during the execution of the query.
- "CPU Speed (cpuspeed)" on page 451.

  The CPU speed is used by the SQL optimizer to estimate the cost of performing certain operations. The optimizer uses these CPU cost estimations along with various I/O cost estimations to select the best access plan for a query.

  The CPU speed of a machine can have a significant influence on the access plan chosen. This configuration parameter is automatically set to an appropriate value when the database is installed or migrated. You should **only** adjust this parameter if you are modelling a production environment on a test system, or to assess the impact of a hardware change. Using this parameter to model a different hardware environment allows you to observe the access plan that will be chosen for that environment.
- "Statement Heap Size (stmtheap)" on page 344.

  The size of the statement heap does not influence the optimizer in choosing different access paths; however, it can affect the amount of optimization that will be performed for complex SQL statements.

  If the *stmtheap* parameter is not set large enough, you may receive an SQL warning indicating that there is not enough memory available to process the statement. For example, SQLCODE +437 (SQLSTATE 01602) can indicate that the amount of optimization that has been used to compile a statement is less than the amount that you requested when you specified the query optimization class. (See "Adjusting the Optimization Class" on page 64 for more information.)
- "Maximum Query Degree of Parallelism (max_querydegree)" on page 443.

When this parameter has a value of "ANY", then the optimizer chooses the degree of parallelism to be used. If other than "ANY" is present, then the user-specified value is used to determine the degree of parallelism for the application.

- "Communications Bandwidth (comm_bandwidth)" on page 450.

  Communications bandwidth is used by the optimizer to determine access paths. The optimizer uses the value in this parameter to estimate the cost of performing certain operations between the database partition servers of a partitioned database.

For additional information, see "Tuning Configuration Parameters" on page 312.

Nodegroup Impact on Query Optimization

In partitioned databases, collocation of tables is recognized by the optimizer and used when determining the best access plan for a query. The assumption is that tables that are frequently involved in join queries should, when divided among partitions in a partitioned database, ideally have the rows from each table being joined located on the same database partition. During the join operation, the collocation of the data from both tables that are part of the join would prevent the need to move data from one partition to another. Placing both tables in the same nodegroup ensures that the data from the tables is collocated together.

Refer to *Administration Guide: Planning* for more information on collocating tables.

Also, within a partitioned database, the spreading of the data over more partitions reduces the estimated time (or cost) to execute a query. The number of tables, the location of the data in those tables, and the type of query (whether a join is required as noted above) all affect the cost of the query.

Table Space Impact on Query Optimization

Certain characteristics of your table spaces can affect the access plan chosen by the SQL compiler:

- Container characteristics

  Container characteristics can have a significant impact on the I/O cost associated when executing a query. When selecting an access plan the SQL optimizer considers these I/O costs, including any cost differences for accessing data from different table spaces. Two columns in the SYSCAT.TABLESPACES system catalog are used by the optimizer to help estimate the I/O costs of accessing data from a table space:

– OVERHEAD, which provides an estimate (in milliseconds) of the time required by the container before any data is read into memory. This overhead activity includes the container's I/O controller overhead as well as the disk latency time, which includes the disk seek time.

  You may use the following formula to help you estimate the overhead cost:

```
OVERHEAD = average seek time in milliseconds
           + (0.5 * rotational latency)
```

  where:
  - 0.5 represents an average overhead of one half rotation
  - Rotational latency is calculated, in milliseconds for each full rotation, as follows:

```
(1 / RPM) * 60 * 1000
```

    where you:
    • Divide by rotations per minute to get minutes per rotation
    • Multiply by 60 seconds per minute
    • Multiply by 1000 milliseconds per second.

  As an example, let the rotations per minute for the disk be 7 200. This would produce, using the rotational latency formula,

```
(1 / 7200) * 60 * 1000 = 8.328 milliseconds
```

  which can then be used in the calculation of the OVERHEAD estimate with an assumed average seek time of 11 milliseconds:

```
OVERHEAD = 11 + (0.5 * 8.328)
         = 15.164
```

  giving an estimated OVERHEAD value of about 15 milliseconds.

– TRANSFERRATE, which provides an estimate (in milliseconds) of the time required to read one page of data into memory.

  If each table space container is a single physical disk then you may use the following formula to help you estimate the transfer cost in milliseconds per page:

```
TRANSFERRATE = (1 / spec_rate) * 1000 / 1 024 000 * page_size
```

  where:
  - `spec_rate` represents the disk specification for the transfer rate, in MB per second
  - Divide by `spec_rate` to get Seconds per MB
  - Multiply by 1000 milliseconds per second
  - Divide by 1 024 000 bytes per MB
  - Multiply by the page size in bytes (for example, 4 096 bytes for a 4 KB page)

As an example, suppose the specification rate for the disk is 3 MB per second. This would produce the following calculation

```
TRANSFERRATE = (1 / 3) * 1000 / 1024000 * 4096
             = 1.333248
```

giving an estimated TRANSFERRATE value of about 1.3 milliseconds per page.

If the table space containers are not single physical disks but rather are arrays of disks (such as RAID), then there are additional considerations when attempting to determine the TRANSFERRATE to use. If the array is relatively small then you can multiply the `spec_rate` by the number of disks, assuming that the bottleneck is at the disk level. However, if the number of disks in the array making up the container is large, then the bottleneck may not be at the disk level, but rather be at one of the other I/O subsystem components such as disk controllers, I/O busses, or the system bus. In this case, you cannot assume that the I/O throughput capability is the product of the `spec_rate` and the number of disks. Instead, you must measure the actual I/O rate (in MBs) during a sequential scan. For example, a sequential scan could be `select count(*) from big_table` and will be MBs in size. Divide the result by the number of containers that make up the table space in which `big_table` resides. Use the result as a substitute for `spec_rate` in the formula given above. For example, a measured sequential I/O rate of 100 MBs while scanning a table in a four container table space would imply 25 MBs per container, or a TRANSFERRATE of (1/25) * 1000 / 1024000 * 4096 = 0.16 milliseconds per page.

Each of the containers assigned to a table space may reside on different physical disks. For best results, all physical disks used for a given table space should have the same OVERHEAD and TRANSFERRATE characteristics. If these characteristics are not the same, you should use the average when setting the values for OVERHEAD and TRANSFERRATE.

You can obtain media specific values for these columns from the hardware specifications or through experimentation. These values may be specified on the CREATE TABLESPACE and ALTER TABLESPACE statements.

Experimentation becomes especially important in the environment mention above where you may have a disk array as a container. You should create a simple query that moves data and use it in conjunction with a platform-specific measuring utility. You can then re-run the query with different container configurations within your table space. You can use the CREATE and ALTER TABLESPACE statements to change how data is transferred in your environment.

The I/O cost information through these two vaules could influence the optimizer in a number of ways, including whether or not to use an index to access the data, and which table to select for the inner and outer tables in a join.

- Prefetching

  When considering the I/O cost of accessing data from a table space, the optimizer will also consider the potential impact that prefetching data and index pages from disk can have on the query performance. Prefetching data and index pages can reduce the overhead and waiting time associated with reading the data into the buffer pool. For more information, see "Prefetching Data into the Buffer Pool" on page 241.

  The optimizer uses the information from the PREFETCHSIZE and EXTENTSIZE columns in SYSCAT.TABLESPACES to estimate the amount of prefetching that will occur for a table space.

  – EXTENTSIZE can only be set when creating a table space (for example using the CREATE TABLESPACE statement). The default extent size is 32 pages (of 4 KB each) and is usually sufficient.

  – PREFETCHSIZE can be set when creating a table space and also using the ALTER TABLESPACE statement. The default prefetch size is determined by the value of the DFT_PREFETCH_SZ database configuration parameter which varies depending on the operating system. You should review the recommendations for sizing this parameter in the "Default Prefetch Size (dft_prefetch_sz)" on page 370 description and make changes as needed to improve the movement of data.

The following shows an example of the syntax to change the characteristics of the RESOURCE table space:

```
ALTER TABLESPACE RESOURCE
  PREFETCHSIZE 64
  OVERHEAD     19.3
  TRANSFERRATE 0.9
```

After making any changes to your table spaces you should consider rebinding your applications and use the RUNSTATS utility to collect the latest statistics about the indexes to ensure the best access plans are being used.

Indexing Impact on Query Optimization

It is important to remember that you do not decide when an index should be used; the database manager makes the decision based on the available table and index information. However, you play an important role in the process by creating the necessary indexes that can improve performance. It is also important for you to collect statistics about the indexes (using the RUNSTATS utility) after you create an index, or change the prefetch size (as mentioned above), and on an ongoing basis to keep the statistics up to date. This means you must understand the kinds of indexes that you can create and the ways to create them.

Indexing versus No Indexing

For each table referenced in a database query, if no index exists on the table, then a table scan must be performed on that table. The larger the table, the longer a table scan takes. A *table scan* occurs when the database manager sequentially accesses every row of a table. This can be compared to an *index scan* that occurs when the database manager accesses data using an index. (See “Index Scan Concepts” on page 153.)

An index will be selected for use if the optimizer estimates that an index scan will be faster than a table scan. Index files generally are smaller and require less time to read than an entire table, particularly as tables grow larger. In addition, the entire index may not need to be scanned. The predicates applied to the index reduce the number of rows to be read from the data pages.

Each index entry consists of a search-key value and a pointer to the row containing that value. The values can be searched in reverse direction only if the ALLOW REVERSE SCANS parameter was specified in the CREATE INDEX statement. It is therefore possible to bracket the search, given the right predicate. An index can also be used to obtain rows in an ordered sequence, eliminating the need for the database manager to sort the rows after they are read from the table. Specifying ALLOW REVERSE SCANS enables the index to be used to directly obtain rows in sequence, in forward and reverse order. Refer to the *SQL Reference* for additional details.

A unique index may contain include columns in addition to the search-key value and row pointer.

Note: You cannot control whether an index is used by the database manager. For example, the result of a query cannot be guaranteed to be produced in an ordered sequence simply by the existence of an index on the table being queried. The database manager may use this index during the processing of the query but is not required to. Only the existence of an ORDER BY clause can “guarantee” the order of a result set.

Indexes can reduce access time significantly; however, indexes can also have adverse effects on performance. Before creating indexes, consider the effects of multiple indexes on disk space and processing time:

- Each index takes up a certain amount of storage or disk space. The exact amount is dependent on the size of the table and the size and number of columns included in the index.

- Each INSERT or DELETE operation performed on a table requires additional updating of each index on that table. This is also true for each UPDATE operation that changes an index key.
- The LOAD utility rebuilds or appends to any existing indexes.
- The `indexfreespace` MODIFIED BY parameter can be specified on the LOAD command to override the index PCTFREE used when the index was created.
- Each index potentially adds an alternative access path for a query, which the optimizer will consider, and therefore increases the query compilation time.

Indexes should be carefully chosen to address the needs of the application program.

To determine whether an index is used in a specific package you may use the SQL Explain facility, described in "Chapter 7. SQL Explain Facility" on page 201.

Using the Index Advisor

The DB2 Index Advisor is a tool to assist you in choosing an optimal set of indexes for your table data. There are different ways to get to this tool:

- You can access this tool through the Control Center by selecting the Indexes folder, clicking mouse button 2, and selecting the **Create —> Index using wizard**.
- You can access this tool from the command line by entering `db2advis`.

More information on the DB2 Index Advisor can be found in "SQL Advise Facility" on page 220.

Guidelines for Indexing

Which indexes should be created depends on the data and its intended uses. The following guidelines can help you determine which indexes would be most useful:

- Define primary keys and unique keys, wherever they apply, by using the CREATE UNIQUE INDEX statement. (Refer to the *SQL Reference* for more information.) Unique indexes can help the optimizer avoid performing certain operations such as sorts.
- Define unique indexes with include columns to improve the performance of data retrieval. Columns are good candidates for INCLUDE columns of unique indexes if they:
  - Are accessed frequently and therefore would benefit from index-only access
  - Are not required to limit the range of index scans
  - Do not affect the ordering or uniqueness of the index key.

Refer to the chapter "Creating an Index or Index Specification" in *Administration Guide: Planning* for more information on INCLUDE columns.

- Use indexes to optimize frequent queries to tables with more than a few data pages, as can be determined by the NPAGES column in the SYSCAT.TABLES catalog view:
  - Create an index on any column you will use when joining tables.
  - Create an index on any column from which you will be searching for particular values on a regular basis.
- Decide between ascending and descending ordering of keys based on which order will be primarily used or requested. The values can be searched in reverse direction only if the ALLOW REVERSE SCANS parameter was specified in the CREATE INDEX statement. Although indexes can be scanned in both forward and reverse directions, a forward scan of the index (that is, in the order specified at the time the index is created) performs slightly better than a reverse scan of the index. Refer to the *SQL Reference* for additional details.
- Avoid creating indexes that are partial keys of other index keys on the columns. For example, if there is an index on columns a, b, and c, then a second index on columns a and b is not generally useful.
- Use indexes on foreign keys to improve performance of delete and update operations on the parent table.
- Use indexes on columns that will frequently be used to sort the data.
- In creating a multiple-column index, if you have more than one choice for the first key column, choose the one most often specified with the "=" predicate or specify the columns with the greatest number of distinct values first.
- Creating indexes, arbitrarily on all columns, not only consumes much disk space, but also causes prepare times to be large. This will be particularly true for complex queries, against which an optimization class with dynamic programming join enumeration is used. (See "Adjusting the Optimization Class" on page 64).
- The following provides a *rule-of-thumb* for the typical number of indexes you will define for a table. This number is based on the primary use of your database:
  - For online transaction processing (OLTP) environments, you should only have one or two indexes
  - For query (read-only) environments, you could have more than five indexes
  - For mixed query/OLTP environments, you could have between two and five indexes.
- Consider defining a clustering index to help keep newly inserted rows clustered according to that index. A clustering index should significantly reduce the need for reorganizing the table.

Note: When a clustering index is defined, the table should be loaded with a free space reserved on each data page to allow inserts to take place on those pages. (Free space is reserved by using the PCTFREE keyword on the ALTER TABLE statement; or, the `pagefreespace` MODIFIED BY clause of the LOAD command.)

- Consider using the PCTFREE keyword when creating indexes. PCTFREE reserves space on index pages for future updates to the index. This may reduce the frequency of page splits and increase performance.
- Consider using the MINPCTUSED option when creating indexes. MINPCTUSED specifies the threshold for the minimum amount of used space on an index leaf page and enables online index reorganization. This could reduce the need for offline reorganization of the data and the index.

Note: Indexes are not supported for declared temporary tables.

The following are typical circumstances in which creating an index can improve performance:

- An index can be created on columns that are used in WHERE clauses of the queries and transactions that are most frequently processed.

  The WHERE clause:

  ```
  WHERE WORKDEPT='A01' OR WORKDEPT='E21'
  ```

  will generally benefit from an index on `WORKDEPT`, unless those values occur frequently.
- An index can be created on a column or columns to order the rows in collating sequence. Ordering is required not only in the ORDER BY clause, but also by other features, such as the DISTINCT and GROUP BY clauses.

  The following example uses the DISTINCT clause:

  ```
  SELECT DISTINCT WORKDEPT
    FROM EMPLOYEE
  ```

  The database manager can use an index defined for ascending or descending order on `WORKDEPT` to eliminate duplicate values. This same index could also be used to group values in the following example with a GROUP BY clause:

  ```
  SELECT WORKDEPT, AVERAGE(SALARY)
    FROM EMPLOYEE
  GROUP BY WORKDEPT
  ```
- An index can be created to name each column that is referenced in a statement. When an index is specified in this way, the resulting index-only access means data can be retrieved more efficiently by avoiding table access.

  For example, assume the following SQL statement is issued:

```
SELECT LASTNAME
  FROM EMPLOYEE
  WHERE WORKDEPT IN ('A00','D11','D21')
```

If an index is defined for the `WORKDEPT` and `LASTNAME` columns of the `EMPLOYEE` table, the statement might be processed more efficiently by scanning the index than by scanning the entire table. Note that since the predicate is on `WORKDEPT`, this column should be the first column of the index.

- Include columns on an index is another way to improve the use of indexes on tables. Using the previous example, you could define a unique index as:

  ```
  CREATE UNIQUE INDEX x ON employee (workdept) INCLUDE (lastname)
  ```

  Specifying `lastname` as an include column rather than as part of the index key means that `lastname` is stored only on the leaf pages of the index.

Performance Tips for Administering Indexes

The following can help you understand how performance can be impacted by properly using and managing indexes:

1. **Index Creation**

   When creating indexes on large tables, and having an SMP machine, consider setting *intra_parallel* to YES (1) or SYSTEM (-1) to take advantage of parallel performance improvements.

   Multiple processors can be used to scan and sort data. The only time when it is not advantageous to have multiple processors during index creation occurs when the *indexsort* database configuration parameter is NO. (The default for the parameter is YES). The parameter controls whether sorting of index keys is done during index creation.

2. **Index Table Space**

   Indexes may be stored in a different table space from that used to store other table data. This can allow for more efficient use of disk storage by reducing the movement of read/write heads. You can also create your index table spaces so they will be stored on faster physical devices.

   A table space may also be assigned a separate buffer pool which may protect the index pages from being pushed out of the buffer by the presence of lots of data pages.

   When indexes are not placed in separate table spaces, both data and index pages use the same extent size and prefetch quantity. If you use a different table space for indexes, you have the option of selecting different values for all the characteristics of a table space. Since indexes are typically smaller than tables and are spread over fewer containers, it is common to find smaller extent sizes such as 8 and 16. For more information see, “Index Page Prefetch” on page 162. Use of faster devices for a table space will be considered by the SQL optimizer, as described in “Table Space

Impact on Query Optimization” on page 90. Refer to *Administration Guide: Planning* for more information about table spaces.

3. **Degree of Clustering**

   If your SQL statement requires ordering (for example, ORDER BY, GROUP BY, DISTINCT) and there is an appropriate index to satisfy the ordering, there may be times that the database manager does *not* choose the index. This could happen when:
   - Index clustering is poor (see the CLUSTERRATIO and CLUSTERFACTOR columns of SYSCAT.INDEXES)
   - The table is small enough that it is cheaper to scan the table and sort the answer set in memory
   - There are competing indexes for accessing the table.

   It is recommended that you perform a REORG, or a sort and LOAD, after creating a clustering index. In general a table can only be clustered on one index. Your tables and indexes should be built in the sequence of the clustering index for that table. A clustering index attempts to maintain a particular order of data, improving the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility.

   You should also consider using PCTFREE when altering a table before loading or reorganizing that table. In order for clustering to be maintained, each table needs to have space available on each data page for additional inserts. When the space is available, additional inserts are able to be clustered with the existing data. As a result, you will want to consider loading your data into the table after leaving a percentage of free space on each page for the clustering of additional data. You can do this by first creating the table, then altering the table with the PCTFREE parameter. In a similar way, before reorganizing your data, you should consider altering the table with the PCTFREE parameter. Otherwise, the reorganization will eliminate all extra space if PCTFREE has not been set.

   Clustering is not currently maintained during updates. That is, if one updates a record such that its key value in the clustering index is changed, the record will not necessarily be moved to a new page to maintain the clustering order. To maintain clustering, instead of using UPDATE, use DELETE and then INSERT.

4. **RUNSTATS Utility**

   After creating a new index, you should use the RUNSTATS utility to collect index statistics. These statistics allow the optimizer to determine whether using the index can improve access performance. See “Collecting Statistics Using the RUNSTATS Utility” on page 108 for more information on this topic.

5. **Reorganizing an Index**

To get the best performance you can from your indexes, you should consider reorganizing your indexes periodically. Updates to your tables may cause index page prefetch to become less effective. To keep the effectiveness of index page prefetch you must reorganize the index.

You can reorganize the index by either dropping and re-creating the index, or by using the REORG utility. For more information, see "Reorganizing Catalogs and User Tables" on page 251.

To prevent having to re-organize often, you can specify PCTFREE when creating an index. Specifying the PCTFREE parameter during index creation results in free space being left on each index leaf page as it is created. As a result, during future activity involving the index, records can be inserted into the index with less likelihood of causing index page splits. Index page splits cause index pages to not be contiguous nor sequential. This results in decreased ability to perform index page prefetching. Choosing an appropriate PCTFREE for an index may eliminate or reduce the frequency when you have to reorganize indexes.

Note: The PCTFREE specified when you create the index is used when the index is re-created during reorganization.

Dropping and re-creating the index gets a new set of pages that are roughly contiguous and sequential. This improves index page prefetch when it occurs.

Although more costly to accomplish, the REORG utility also ensures clustering of the data pages. This clustering has greater benefit for index scans accessing a significant number of data pages.

If you work in a symmetric multi-processor (SMP) system environment, the REORG utility will use multiple processors when *intra_parallel* is YES or ANY.

6. **Use EXPLAIN**

   Periodically, run EXPLAIN on your most frequently used queries and check that each of your indexes is used at least once. If an index is not used in any query, consider dropping that index.

   Also, use EXPLAIN to see if table scans on large tables are processed as the inner of nested loop joins. This would indicate that an index on the join predicate column is either missing or thought to be ineffective at applying the join predicate. Or, perhaps the join predicate is not present.

7. **Volatile Tables**

   A *volatile* table is defined as a table whose contents can vary from empty to very large at run time. Generating an access plan that uses a volatile table can result in the optimizer favoring the use of a table scan rather than an index scan to access the volatile table.

Declaring a table "volatile" using the ALTER TABLE...VOLATILE statement can allow the optimizer to use an index scan on the volatile table. Refer to *Administration Guide: Planning* or the *SQL Reference* for additional information on this topic.

Server Options Affecting Federated Database Queries

A federated system is composed of a DB2 DBMS (the federated database) and one or more data sources. Data sources are identified to the federated database when you issue CREATE SERVER statements. When you issue these statements, you can also provide server options that refine and control aspects of federated system operations involving DB2 and the specified data source. Server options can be changed later using ALTER SERVER statements. Refer to the *SQL Reference* for more information about the CREATE SERVER and ALTER SERVER statements.

Note: You must install the distributed join installation option and set the database manager parameter FEDERATED to YES before you can create servers and specify server options.

Server options and their values facilitate query pushdown analysis, global optimization and other aspects of federated database operations. For example: in the CREATE SERVER statement, you can specify certain performance statistics as server option values. That is, you can set the *cpu_ratio* option to a value that indicates the relative speeds of the data source's and federated server's CPUs. And you can set the *io_ratio* option to a value that indicates the relative rates of the data source's and federated server's I/O devices. When you run CREATE SERVER, this data is added to the catalog view SYSCAT.SERVEROPTIONS, and the optimizer uses it in developing its access plan for the data source. If a statistic changes (as might happen, for instance, if the data source CPU is upgraded), you can use the ALTER SERVER statement to update SYSCAT.SERVEROPTIONS with this change. The optimizer then uses your update in developing its next access plan for the data source.

Table 8. Server Options and Their Settings

Option	Valid Settings	Default Setting
collating_sequence	Specifies whether the data source uses the same default collating sequence as the federated database, based on the code set and the country information. If a data source has a collating sequence that differs from DB2's collating sequence, most operations depending on DB2's collating sequence cannot be remotely evaluated at a data source. An example is executing MAX column functions against a nickname character column at a data source with a different collating sequence. Because results might differ if the MAX function is evaluated at the remote data source, DB2 will perform the aggregate operation and the MAX function locally. If your query contains an equal sign, it is possible to push-down that portion of the query even if the collating sequences are different (set to 'N'). For example, the predicate C1 = 'A' could be pushed-down to a data source. Of course, such queries cannot be pushed-down when the collating sequence at the data source is case-insensitive. When a data source is case-insensitive, the results from C1= 'A' and C1 = 'a' are the same, which is not acceptable in a case-sensitive environment (DB2). Administrators can create federated databases with a particular collating sequence that matches the data source collating sequence. This approach may speed performance if all data sources use the same collating sequence or if most or all column functions are directed against data sources that use the same collating sequence. 'Y' Data source's collating sequence is the same as federated database's. 'N' Data source's collating sequence is not the same as federated database's. 'I' Data source's collating sequence is different from federated database's and is case-insensitive (for example, 'TOLLESON' and 'TolLESon' are considered equal).	'N'
comm_rate	Specifies the communication rate between a federated server and its associated data sources. Expressed in megabytes per second. Valid values are greater than 0 and less than 2147483648. Values may be expressed as whole numbers only, for example 12.	'2'

Table 8. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
connectstring	Specifies initialization properties needed to connect to an OLE DB provider. For the complete syntax and semantics of the connection string, see the ″Data Link API of the OLE DB Core Components″ in the *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK, Microsoft Press, 1998*.	None
cpu_ratio	Indicates how much faster or slower a data source's CPU runs than the federated server's CPU. Valid values are greater than 0 and less than $1x10^{23}$. Values may be expressed in any valid double notation, for example 123E10, 123, or 1.21E4.	′1.0′
dbname	Name of the data source database that you want the federated server to access. Required for DB2 family data sources; does not apply to Oracle** data sources because Oracle instances contain only one database. For DB2, this value corresponds to a specific database within an instance or, if DB2 for OS/390, the database LOCATION value.	None.
fold_id (See notes 1 and 4 at the end of this table.)	Applies to user IDs that the federated server sends to data sources for authentication. Valid values are: ′U′ The federated server folds the user ID to uppercase before sending it to the data source. This is a logical choice for DB2 family and Oracle** data sources (See note 2 at end of this table.) ′N′ The federated server does nothing to the user ID before sending it to the data source. (See note 2 at end of this table.) ′L′ The federated server folds the user ID to lowercase before sending it to the data source. If none of these settings are used, the federated server tries to send the user ID to the data source in uppercase. If the user ID fails, the server tries sending it in lowercase.	None.

Table 8. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
fold_pw (See notes 1, 3 and 4 at the end of this table.)	Applies to passwords that the federated server sends to data sources for authentication. Valid values are: 'U' The federated server folds the password to uppercase before sending it to the data source. This is a logical choice for DB2 family and Oracle** data sources. 'N' The federated server does nothing to the password before sending it to the data source. 'L' The federated server folds the password to lowercase before sending it to the data source. If none of these settings are used, the federated server tries to send the password to the data source in uppercase. If the password fails, the server tries sending it in lowercase.	None.
io_ratio	Denotes how much faster or slower a data source's I/O system runs than the federated server's I/O system. Valid values are greater than 0 and less than $1x10^{23}$. Values may be expressed in any valid double notation, for example 123E10, 123, or 1.21E4.	'1.0'
node	Name by which a data source is defined as an instance to its RDBMS. Required for all data sources. For a DB2 family data source, this name is the node specified in the federated database's `DB2 node` directory. To view this directory, issue the **db2 list node directory** command. For an Oracle** data source, this name is the server name specified in the Oracle** `tnsnames.ora` file. To access this name on the Windows NT platform, specify the **View Configuration Information** option of the Oracle** SQL Net Easy Configuration tool.	None.
password	Specifies whether passwords are sent to a data source. 'Y' Passwords are always sent to the data source and validated. This is the default value. 'N' Passwords are not sent to the data source (regardless of any user mappings) and not validated. 'ENCRYPTION' Passwords are always sent to the data source in encrypted form and validated. Valid only for DB2 family data sources that support encrypted passwords.	'Y'

Table 8. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
plan_hints	Specifies whether *plan hints* are to be enabled. Plan hints are statement fragments that provide extra information for data source optimizers. This information can, for certain query types, improve query performance. The plan hints can help the data source optimizer decide whether to use an index, which index to use, or which table join sequence to use. ’Y’ Plan hints are to be enabled at the data source if the data source supports plan hints. ’N’ Plan hints are not to be enabled at the data source.	’N’
pushdown	’Y’ DB2 will consider letting the data source evaluate operations. ’N’ DB2 will retrieve only columns from the remote data source and will not let the data source evaluate other operations, such as joins.	’Y’
varchar_no_trailing_blanks	Specifies if this data source uses non-blank padded varchar comparison semantics. For varying-length character strings that contain no trailing blanks, some DBMS’ s non-blank-padded comparison semantics return the same results as DB2’s comparison semantics. If you are certain that all VARCHAR table/view columns at a data source contain no trailing blanks, consider setting this server option to ’Y’ for a data source. This option is often used with Oracle** data sources. Ensure that you consider all objects that can potentially have nicknames (including views). ’Y’ This data source has non-blank-padded comparison semantics similar to DB2’s. ’N’ This data source does not have the same non-blank-padded comparison semantics as DB2’s.	’N’

Notes on Table 8 on page 102:

1. This field is applied regardless of the value specified for authentication.
2. Because DB2 stores user IDs in uppercase, the values ‘N’ and ‘U’ are logically equivalent to each other.
3. The setting for fold_pw has no effect when the setting for password is ‘N’. Because no password is sent, case cannot be a factor.

4. Avoid null settings for either of these options. A null setting may seem attractive because DB2 will make multiple attempts to resolve user IDs and passwords; however, performance might suffer (it is possible that DB2 will send a user ID and password four times before successfully passing data source authentication).

Chapter 5. System Catalog Statistics

When optimizing SQL queries, the decisions made by the SQL compiler are heavily influenced by the optimizer's model of the database contents. This data model is used by the optimizer to estimate the costs of alternative access paths that could be used to resolve a particular query.

A key element in the data model is the set of statistics gathered about the data contained in the database and stored in the system catalog tables. This includes statistics for tables, nicknames, indexes, columns, and user-defined functions (UDFs). A change in the data statistics can result in a change in the choice of access plan selected as the most efficient method of accessing the desired data.

Examples of the statistics available which help define the data model to the optimizer include:
- The number of pages in a table and the number of pages that are not empty
- The degree to which rows have been moved from their original page to other (overflow) pages.
- The number of rows in a table
- The number of distinct values in a column
- The degree of clustering of an index. That is, the extent to which the physical sequence of rows in a table follows an index.
- The number of index levels and the number of leaf pages in each index
- The number of occurrences of frequently used column values (see "Collecting and Using Distribution Statistics" on page 116)
- The distribution of column values across the range of values present in the column (see "Collecting and Using Distribution Statistics" on page 116)
- Cost estimates for user-defined functions (UDFs).

Statistics for objects are updated in the system catalog tables only when explicitly requested. Some or all of the statistics may be updated by:
- Using the RUNSTATS (run statistics) utility (see "Collecting Statistics Using the RUNSTATS Utility" on page 108)
- Using LOAD, with statistics collection options specified
- Coding SQL UPDATE statements that operate against a set of predefined catalog views (see "User Update-Capable Catalog Statistics" on page 128). Note that statistics for user-defined functions must be updated using this technique (see "Updating Statistics for User-Defined Functions" on page 134). Except for UDFs, the catalogs should only be updated manually for modeling a production environment on a test system or for "what-if analysis". Statistics should not be updated on production systems.

Within a federated database system, the only way to gather new statistics for nicknames from the data source is to drop the nickname, run the equivalent of RUNSTATS at the data source, and then re-create the nickname. Whenever a nickname is created, statistics on the underlying table are gathered from the data source catalog.

You must drop and then re-create nicknames if the data definition information in the underlying table changes. For example, if a column is added to a table definition.

In addition you should consider re-creating the nickname if query performance degrades. Another approach is to manually update statistics in the SYSSTAT.TABLES.

Use caution when creating a nickname for a view. The statistical information, such as the number of rows this nickname will return, might not reflect the real cost to evaluate this view. If the view is defined on a single base table with no column functions applied on the SELECT list, the statistical information available to the optimizer should be accurate. If the view is complex, consider creating new views over nicknames for the view base tables at the DB2 Universal Database server in the federated database system so the optimizer can generate an efficient plan to access the data.

Additional Information:

The SYSCAT and SYSSTAT catalogs contain information on the statistics gathered. Refer to the *SQL Reference*:

- For information about all the catalog views and the columns they contain.
- For information about all the update-capable catalog views and the columns they contain. You can also refer to this section if you are only interested in the statistical columns of the catalog table.
- For information about table statistics.
- For information about column statistics.
- For information about column distribution statistics.
- For information about index statistics.
- For information about user-defined function statistics.

Collecting Statistics Using the RUNSTATS Utility

The RUNSTATS utility updates statistics in the system catalog tables to help with the query optimization process. Without these statistics, the database manager could make a decision that would adversely affect the performance of an SQL statement. The RUNSTATS utility allows you to collect statistics on the data contained in the tables, indexes, or both tables and indexes.

Use the RUNSTATS utility to collect statistics based on both the table and the index data to provide accurate information to the access plan selection process in the following situations:
- When a table has been loaded with data, and the appropriate indexes have been created.
- When a table has been reorganized with the REORG utility.
- When there have been extensive updates, deletions, and insertions that affect a table and its indexes. (“Extensive” in this case may mean that 10 to 20 percent of the table and index data has been affected.)
- Before binding application programs whose performance is critical
- When comparison with previous statistics is desired. Running statistics on a periodic basis permits the discovery of performance problems at an early stage, as described below.
- When the prefetch quantity is changed.
- When you have used the REDISTRIBUTE NODEGROUP utility.

When you are working in a partitioned database, collect the statistics related to a table and its indexes by executing the RUNSTATS operation at a single node. (The node at which the utility executes is determined by whether the node at which you issue the command contains table data or not. See “The Database Partition Where RUNSTATS is Executed” for details.) Because the statistics stored in the catalogs are supposed to represent table-level information, the node-level statistics collected by the database manager are multiplied where appropriate by the number of nodes across which the table is partitioned. This provides an approximation of the actual statistics that would be collected by executing RUNSTATS at every node and aggregating these statistics.

Note: The DB2 query optimizer assumes that attribute values (data) are placed equally and evenly across the database partitions of the system. If the placement of data is not equal, you should run this command on a database partition that you think has a representative table distribution.

The Database Partition Where RUNSTATS is Executed

When you invoke RUNSTATS on a table, you must be connected to the database in which the table is stored, but the database partition from which you issue the command does not have to contain a partition for this table:

- If you issue RUNSTATS from a database partition that contains a partition for the table, the utility executes at this database partition.
- If you issue RUNSTATS from a database partition that does not contain a table partition, the request is sent to the first database partition in the nodegroup that holds a partition for the table. The utility then executes at this database partition.

Analyzing Statistics

Analyzing the statistics can indicate when reorganization is necessary. Some of these indications are:

- Clustering of indexes

  If cluster ratio statistics are collected, their value will be in the range from 0 to 100. If cluster factor statistics are collected, their value will be a number between 0 and 1. Only one of these two clustering statistics will be recorded in the SYSCAT.INDEXES catalog. In general, only one of the indexes in a table can have a high degree of clustering. A value of -1 is used to indicate that no statistics are available.

  If you wish to compare ratio values, multiply the cluster factor by 100 to obtain a percentage value for the amount of clustering.

  Index scans that are **not** index-only accesses might perform better with higher cluster ratios. A low cluster ratio leads to more I/O for this type of scan, since after the first access of each data page, it is less likely that the page is still in the buffer pool the next time it is accessed. Increasing the buffer size can improve the performance of an unclustered index. (See "Understanding List Prefetching" on page 243 for information about how the database manager can improve index scan performance for indexes with low cluster ratios and see "Clustered Indexes" on page 160 for information about how the optimizer uses index statistics.)

  If the table data was initially clustered with respect to a certain index, and the above clustering information indicates that the data is now poorly clustered for that same index, you may wish to reorganize the table to re-cluster the data with respect to that index.

- Overflow of rows

  The overflow number indicates the number of rows that do not fit on their original pages. This can occur when VARCHAR columns are updated with longer values. In such cases, a pointer is kept at the row's original location. This can hurt performance, because the database manager must follow the pointer to find the row's contents, which increases the processing time and may also increase the number of I/Os.

  As the number of overflow rows grows higher, the potential benefit of reorganizing your table data also increases. Reorganizing the table data will eliminate the overflowing of rows.

- Comparison of file pages

  The number of pages with rows can be compared with the total number of pages that a table contains. Empty pages will be read for a table scan. Empty pages can occur when entire ranges of rows are deleted.

  As the number of empty pages grows higher, so does the need for a table reorganization. Reorganizing the table can compress the amount of space used by a table, by reclaiming these empty pages. In addition to more efficient use of disk space, reclaiming unused pages can also improve the performance of a table scan, since fewer pages will be read into the buffer pool.

- Number of leaf pages

  The number of leaf pages predicts how many index page I/Os are needed for a complete scan of an index.

  Random update activity can cause page splits to occur that increase the size of the index beyond the minimum amount of space required. When indexes are rebuilt during the reorganization of a table, it is possible to build each index with the minimum amount of space possible. For more information on the minimum space requirements for an index, see “Indexing Impact on Query Optimization” on page 93 or refer to “Creating an Index or an Index Specification” section in the *Administration Guide: Planning*.

  Note: A default of ten percent free space is left on each index page when the indexes are rebuilt. You can increase the free space amount by using the PCTFREE parameter when first creating the index. Then, whenever you reorganize the index, the PCTFREE value is used. Having a free space larger than ten percent may be important if you wish to reduce the number of times you need to reorganize the index. The free space is used to accommodate additional index inserts.

RUNSTATS can also help you determine how performance is related to changes in your database. The statistics show the data distribution within a table. When used routinely, RUNSTATS provides data about tables and indexes over a period of time, thereby allowing performance trends to be identified for your data model as it evolves over time.

Ideally, you should rebind application programs after running statistics, because the query optimizer may choose a different access plan given the new statistics.

If you do not have enough time available to collect all of the statistics at one time, you may choose to periodically run RUNSTATS to update only a portion of the statistics that could be gathered. If inconsistencies are found as a result of activity on the table between the periods where you run RUNSTATS with a selective partial update, then a warning message (SQL0437W, reason code 6) is issued. For example, you first use RUNSTATS to gather table distribution statistics. Subsequently, you use RUNSTATS to gather index statistics. If inconsistencies are detected as a result of activity on the table, then the table distribution statistics are dropped and the warning message is issued. It is recommended that you run RUNSTATS to gather table distribution statistics when this happens.

You should periodically use RUNSTATS to gather both table and index statistics at once, to ensure that the index statistics are synchronized with the table statistics. Index statistics retain most of the table and column statistics collected from the last run of RUNSTATS. If the table has been modified extensively since the last time its table statistics were gathered, gathering only the index statistics for that table will leave the two sets of statistics out of synchronization.

You may wish to collect statistics based only on index data in the following situations:
- A new index has been created since the utility was performed and you do not want to re-collect statistics on the table data.
- There have been a lot of changes to the data that affect the first column of an index.

The RUNSTATS utility allows you to collect varying levels of statistics. For tables, you can collect basic level statistics or you can also collect distribution statistics for the column values within a table (see "Collecting and Using Distribution Statistics" on page 116). For indexes, you can collect basic level statistics or you can also collect detailed statistics which can help the optimizer better estimate the I/O cost of an index scan. (See "Clustered Indexes" on page 160 for information about these "detailed" statistics).

Note: Statistics are not collected for LONG, large object (LOB), or structured type columns. For row types, the table level statistics NPAGES, FPAGES, and OVERFLOW are not collected for a sub-table. Statistics are not collected for extended indexes, nor for declared temporary tables.

The following tables show the catalog statistics that are updated by the RUNSTATS utility:

Table 9. Table Statistics (SYSCAT.TABLES and SYSSTAT.TABLES)

Statistic	**Description**	**RUNSTATS Option**	
		Table	**Indexes**
FPAGES	number of pages being used by a table	Yes	Yes
NPAGES	number of pages containing rows	Yes	Yes
OVERFLOW	number of rows that overflow	Yes	No
CARD	number of rows in table (cardinality)	Yes	Yes (Note 2)

Table 9. Table Statistics (SYSCAT.TABLES and SYSSTAT.TABLES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes

Note:
1. For a partitioned database, the values for each statistic are estimated from the value of the count at the database partition multiplied by the number of database partitions.
2. If the table has no indices defined and you request statistics for indexes, no new CARD statistics are updated. The previous CARD statistics are retained.

Table 10. Column Statistics (SYSCAT.COLUMNS and SYSSTAT.COLUMNS)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
COLCARD	column cardinality	Yes (Note 1)	Yes (Note 2)
AVGCOLLEN	average length of column	Yes	Yes (Note 2)
HIGH2KEY	second highest value in column	Yes	Yes (Note 2)
LOW2KEY	second lowest value in column	Yes	Yes (Note 2)
NUMNULLS	the number of NULLs in a column	Yes	Yes (Note 2)

Note:
1. COLCARD is estimated for all columns in the table. In a partitioned database, if the column is the single-column partitioning key for the table, the value of the count is estimated as the count at the database partition multiplied by the number of database partitions.
2. Column statistics are gathered for the first column in the index key.

Table 11. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
NLEAF	number of index leaf pages	No	Yes (Note 3)
NLEVELS	number of index levels	No	Yes
CLUSTERRATIO	degree of clustering of table data	No	Yes (Note 2)
CLUSTERFACTOR	finer degree of clustering	No	Detailed (Notes 1,2)

Table 11. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES) (continued)

Statistic	**Description**	**RUNSTATS Option**	
		Table	**Indexes**
DENSITY	Ratio (percentage) of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index (Note 4)	No	Yes
FIRSTKEYCARD	number of distinct values in first column of the index	No	Yes (Note 3)
FIRST2KEYCARD	number of distinct values in first two columns of the index	No	Yes (Note 3)
FIRST3KEYCARD	number of distinct values in first three columns of the index	No	Yes (Note 3)
FIRST4KEYCARD	number of distinct values in first four columns of the index	No	Yes (Note 3)
FULLKEYCARD	number of distinct values in all columns of the index	No	Yes (Note 3)
PAGE_FETCH_PAIRS	page fetch estimates for different buffer sizes	No	Detailed (Notes 1,2)
SEQUENTIAL_PAGES	number of leaf pages located on disk in index key order, with few or no large gaps between them	No	Yes

Table 11. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes

Note:

1. Detailed index statistics are gathered by specifying the DETAILED clause on the RUNSTATS command, or by specifying `A`, `Y` or `X` for the `statsopt` parameter when calling the RUNSTATS API.
2. CLUSTER_FACTOR and PAGE_FETCH_PAIRS are not collected with the DETAILED clause unless the table is of a respectable size. If the table is greater than about 25 pages, then CLUSTERFACTOR and PAGE_FETCH_PAIRS statistics are collected. In this case, CLUSTERRATIO is -1 (not collected). If the table is a relatively small table, only CLUSTERRATIO is filled in by RUNSTATS while CLUSTERFACTOR and PAGE_FETCH_PAIRS are not. If the DETAILED clause is not specified, only the CLUSTERRATIO statistic is collected.
3. For a partitioned database, the value is estimated from the value of the count at the database partition multiplied by the number of database partitions.
4. This statistic measures the percentage of pages in the file containing the index that belongs to that table. For a table having only one index defined on it, DENSITY should normally be 100. DENSITY is used by the optimizer to estimate how many irrelevant pages from other indexes might be read, on average, if the index pages were prefetched.

Table 12. Column Distribution Statistics (SYSCAT.COLDIST and SYSSTAT.COLDIST)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
DISTCOUNT	If TYPE is Q, the number of distinct values that are less than or equal to COLVALUE statistics	Distribution (Note 2)	No
TYPE	Indicator of whether row provides frequent-value or quantile statistics	Distribution	No
SEQNO	Frequency ranking of a sequence number to help uniquely identify the row in the table	Distribution	No
COLVALUE	Data value for which frequency or quantile statistic is collected	Distribution	No
VALCOUNT	Frequency with which the data value occurs in column, or for quantiles, the number of values less than or equal to the data value (COLVALUE)	Distribution	No

Table 12. Column Distribution Statistics (SYSCAT.COLDIST and SYSSTAT.COLDIST) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
Note: 1. Column distribution statistics are gathered by specifying the WITH DISTRIBUTION clause on the RUNSTATS command, or by specifying `A`, `D` or `Y` for the `statsopt` parameter when calling the RUNSTATS API. Note that distribution statistics may **not** be gathered unless there is a sufficient lack of uniformity in the column values. 2. DISTCOUNT is collected only for columns that are the first key column in an index. 3. In a partitioned database, VALCOUNT is the estimated value of the count at the database partition multiplied by the number of database partitions. The exception to this is where the TYPE is 'F' and the column is the single-column partitioning key of the table, in which case VALCOUNT is simply the count at the database partition.			

For more information about column distribution statistics, see "Collecting and Using Distribution Statistics".

Statistics for user-defined functions are not collected by the RUNSTATS utility. You must manually update the statistics for these functions. See "User Update-Capable Catalog Statistics" on page 128 and "Updating Statistics for User-Defined Functions" on page 134.

Collecting and Using Distribution Statistics

The database manager can collect, maintain, and use "frequent-value statistics" and "quantiles", two types of statistics that estimate, in a concise way, the distribution of the data values in a column. Use of these statistics by the optimizer can lead to significantly more accurate estimates of the number of rows in a column that satisfy given equality or range predicates. These more accurate estimates in turn increase the likelihood that the optimizer will choose an optimal plan.

You may collect statistics about the distribution of these data values by using the WITH DISTRIBUTION clause on the RUNSTATS command. While collecting these additional statistics results in additional overhead for the RUNSTATS utility, the SQL compiler can use this information to help ensure the best access plan is chosen.

In some cases, the database manager will not collect distribution statistics and no error will be returned. For example:

- The *num_freqvalues* and *num_quantiles* configuration parameters are set to zero (0) to indicate that you do not want to collect distribution statistics. For more information about these parameters, see:
  - "How Many Statistics Should You Keep?" on page 120
  - "Number of Frequent Values Retained (num_freqvalues)" on page 422

  – “Number of Quantiles for Columns (num_quantiles)” on page 423.
- The distribution of the data is known without the use of distribution statistics. For example, a column that does not have any data value appearing more than once, that is, each data value in the column is unique.
- The data type is one for which statistics are not collected. That is, the column is defined using a long field or large object data type.
- In the case of quantiles, there is only one non-NULL value in the column.

Distribution statistics are exact for the first column of indexes. For each additional column, the database manager uses hashing and sampling techniques to estimate the distribution statistics because calculating exact statistics would require too much time and memory to be practical. These techniques are accepted statistical methods with accepted degrees of accuracy.

Distribution statistics can be removed by updating SYSSTAT.COLDIST and setting all the COLVALUE and VALCOUNT values to either 0 or -1 for the columns for which distribution statistics are no longer needed.

The following topics provide information to help you understand and use these distribution statistics:
- Understanding Distribution Statistics.
- When Should You Use Distribution Statistics?
- How Many Statistics Should You Keep?
- How Does the Optimizer Use Distribution Statistics?
- Modeling Production Databases.
- Rules for Updating Distribution Statistics for Columns.

Understanding Distribution Statistics

For a fixed number N>=1, the *N most frequent values* in a column consist of the data value having the highest frequency (that is, number of duplicates), the data value having the second highest frequency, and so forth, down to the data value having the Nth highest frequency. The corresponding *frequent-value statistics* consist of these “N” data values, together with the frequencies of these values in the column.

The *K-quantile* for a column is the smallest data value, `V`, such that at least “K” rows have data values less than or equal to `V`. A K-quantile can be computed by sorting the rows in the column according to increasing data values; the K-quantile is the data value in the Kth row of the sorted column.

For example, consider the following column of data:

```
C1
--
 B
 E
 Y
 B
 F
 G
 E
 A
 J
 K
 E
 L
```

This column can be sorted to obtain the following ordered values:

```
C1'
--
 A
 B
 B
 E
 E
 E
 F
 G
 J
 K
 L
 Y
```

There are nine distinct data values in column C1. For N = 2, the frequent value statistics are:

```
SEQNO    COLVALUE     VALCOUNT
-----    ---------    --------
  1          E            3
  2          B            2
```

If the number of quantiles being collected is 5 (see “Number of Quantiles for Columns (num_quantiles)” on page 423), then the K-quantiles for this column for K = 1, 3, 6, 9, and 12 are:

```
SEQNO    COLVALUE     VALCOUNT
-----    ---------    --------
  1          A            1
  2          B            3
  3          E            6
  4          J            9
  5          Y           12
```

In this example, the 6-quantile is equal to E since the sixth row in the sorted column has a data value equal to E (and 6 rows in the original column have data values less than or equal to E).

The same quantile value may occur more than once, if it is a common value. A maximum of two quantiles will be stored for a given value. The first of these two quantiles has a COLCOUNT that gives the number of rows strictly less than COLVALUE, and the second of the two quantiles gives the number of rows less than or equal to COLVALUE.

When Should You Use Distribution Statistics?

To decide whether distribution statistics should be kept for a given table, two factors should be considered:

1. The use of static or dynamic SQL.

   Distribution statistics are most useful for dynamic SQL and static SQL that does not use host variables. When using SQL with host variables, the optimizer makes limited use of distribution statistics.
2. The lack of uniformity in the data distributions.

   Keeping distribution statistics is advisable if at least one column in the table has a highly "non-uniform" distribution of data and the column appears frequently in equality or range predicates; that is, in clauses such as the following:

   ```
   WHERE C1 = KEY;
   WHERE C1 IN (KEY1, KEY2, KEY3);
   WHERE (C1 = KEY1) OR (C1 = KEY2) OR (C1 = KEY3);
   WHERE C1 <= KEY;
   WHERE C1 BETWEEN KEY1 AND KEY2;
   ```

   There can be two types of non-uniformity in a data distribution, possibly occurring together:
   - One type of non-uniformity occurs when the data, instead of being evenly spread out between the highest and lowest data value, is clustered in some sub-interval, as in the following column, where the data is clustered in the range (5,10):

     ```
       C1
     -----
       0.0
       5.1
       6.3
       7.1
       8.2
       8.4
       8.5
       9.1
      93.6
     100.0
     ```

     It can be useful to keep quantiles when this type of non-uniformity is present.

     The following example shows a query that can be used to help determine whether a high degree of non-uniformity exists in a column.

```
SELECT C1, COUNT(*) AS OCCURRENCES
  FROM T1
GROUP BY C1
ORDER BY OCCURRENCES DESC;
```

- Another type of non-uniformity occurs when certain data values have a much higher frequency than other data values, as in a column having data values with the following frequencies:

  ```
  Data Value   Frequency
  ----------   ---------
      20           5
      30          10
      40          10
      50          25
      60          25
      70          20
      80           5
  ```

  It can be useful to keep both quantiles and frequent-value statistics when this type of non-uniformity is present.

You may collect distribution statistics by using the WITH DISTRIBUTION clause on the RUNSTATS command; or by specifying `D`, `E`, or `A` for the `statsopt` parameter when calling the RUNSTATS API. For more information on the application programming interface, refer to the *Administrative API Reference* manual.

How Many Statistics Should You Keep?

Keeping a large number of column distribution statistics can lead to improved selection of access plans by the optimizer, but the cost of collecting these statistics and compiling your queries increases accordingly. The size of the statistics heap (see "Statistics Heap Size (stat_heap_sz)" on page 345) may place limitations on the number of statistics that can be computed and stored.

When distribution statistics are requested, the database manager stores a default of the 10 most frequent values for a column. Keeping between 10 and 100 frequent values should suffice for most practical situations. Ideally, enough frequent-value statistics should be retained so that the frequencies of the remaining values are either approximately equal to each other or negligible compared to the frequencies of the most frequent values.

To set the number of frequent values to collect, use the *num_freqvalues* configuration parameter, as described in "Number of Frequent Values Retained (num_freqvalues)" on page 422. The database manager may collect less than this number of frequent value statistics, because these statistics will only be collected for data values that occur more than once. If collecting only quantile statistics, this parameter can be set to zero.

When distribution statistics are requested, the database manager stores a default of 20 quantiles for a column. This value guarantees a maximum estimation error of approximately 2.5% for any simple single-sided range predicate (>, >=, <, or <=), and a maximum error of 5% for any BETWEEN predicate. A rough rule of thumb for determining the number of quantiles is:

- Determine the maximum error that is tolerable in estimating the number of rows of any range query, as a percentage, P
- The number of quantiles should be approximately 100/P if the predicate is a BETWEEN predicate, and 50/P if the predicate is any other type of range predicate (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and of 2% for ″>″ predicates. In general, at least 10 quantiles should be kept, and more than 50 quantiles should be necessary only for extremely non-uniform data.

To set the number of quantiles, use the *num_quantiles* configuration parameter as described in “Number of Quantiles for Columns (num_quantiles)” on page 423. If collecting only frequent value statistics, this parameter can be set to zero. Setting this parameter to “1” will also result in no quantile statistics being gathered since the entire range of values will fit in one quantile.

How Does the Optimizer Use Distribution Statistics?

Why collect and store distribution statistics? The answer lies in the fact that an optimizer needs to estimate the number of rows in a column that satisfy an equality or range predicate in order to select the least expensive access plan. The more accurate the estimate, the greater the likelihood that the optimizer will choose the optimal access plan. For example, consider the query

```
SELECT C1, C2
  FROM TABLE1
  WHERE C1 = 'NEW YORK'
  AND C2 <= 10
```

and suppose that there is an index on C1 and an index on C2. One possible access plan is to use the index on C1 to retrieve all rows with `C1 = 'NEW YORK'` and then check each retrieved row to see if `C2 <= 10`. An alternative plan is to use the index on C2 to retrieve all rows with `C2 <= 10` and then check each retrieved row to see if `C1 = 'NEW YORK'`. Typically, the primary cost in executing the above query is the cost of the retrieving the rows, and so it is desirable to choose the plan the that requires the minimum number of retrievals. To choose the best plan, it is necessary to estimate the number of rows that satisfy each predicate.

When you do not request distribution statistics, the optimizer maintains only the second-highest data value (HIGH2KEY), second-lowest data value (LOW2KEY), number of distinct values (COLCARD), and number of rows (CARD) for a column. The number of rows that satisfy an equality or range predicate is then estimated under the assumption that the frequencies of the data values in a column are all equal and the data values are evenly spread out over the interval (LOW2KEY, HIGH2KEY). Specifically, the number of rows satisfying an equality predicate `C1 = KEY` is estimated as CARD/COLCARD, and the number of rows satisfying a range predicate `C1 BETWEEN KEY1 AND KEY2` is estimated as:

```
       KEY2 - KEY1
   -------------------  x CARD       (1)
    HIGH2KEY - LOW2KEY
```

These estimates are accurate only when the true distribution of data values in a column is reasonably uniform. When distribution statistics are unavailable and either the frequencies of the data values differ widely from each other or the data values are clustered in a few sub-intervals of the interval (LOW_KEY,HIGH_KEY), the estimates can be off by orders of magnitude and the optimizer may choose a less than optimal access plan.

When distribution statistics are available, the errors described above can be greatly reduced by using frequent-value statistics to compute the number of rows that satisfy an equality predicate and using frequent-value statistics and quantiles to compute the number of rows that satisfy a range predicate.

Example of Impact on Equality Predicates:

Consider first a predicate of the form `C1 = KEY`. If `KEY` is one of the N most frequent values, then the optimizer simply uses the frequency of `KEY` that is stored in the catalog. If `KEY` is not one of the N most frequent values, the optimizer estimates the number of rows that satisfy the predicate under the assumption that the (COLCARD - N) non-frequent values have a uniform distribution. That is, the number of rows is estimated as:

```
   CARD - NUM_FREQ_ROWS
   --------------------           (2)
      COLCARD - N
```

where NUM_FREQ_ROWS is the total number of rows with a value equal to one of the N most frequent values.

For example, consider a column (`C`) for which the frequency of the data values is as follows:

```
   Data Value   Frequency
   ----------   ---------
       1            2
       2            3
       3           40
       4            4
       5            1
```

Suppose that frequent-value statistics based on only the most frequent value (that is, N = 1) are available. For this column, CARD = 50 and COLCARD = 5. For the predicate `C = 3`, exactly 40 rows satisfy it. Assuming a uniform data distribution, the number of rows that satisfy the predicate is estimated as 50/5 = 10, an error of -75%. Using frequent-value statistics, the number of rows is estimated as 40, with no error.

Similarly, 2 rows satisfy the predicate `C = 1`. Without frequent-value statistics, the number of rows that satisfy the predicate is estimated as 10, an error of 400%. You may use the following formula to calculate the estimation error (as a percentage):

```
estimated rows  - actual rows
-----------------------------  X 100
          actual rows
```

Using the frequent value statistics (N = 1), the optimizer will estimate the number of rows containing this value using the formula (2) given above, for example:

```
(50 - 40)
--------- = 3
 (5 - 1)
```

and the error is reduced by an order of magnitude as shown below:

```
 3 - 2
------- = 50%
   2
```

The number of rows that satisfy a range predicate can be estimated using quantiles, as illustrated by the following examples. Consider a column (`C`) given by:

```
    C
 -------
    0.0
    5.1
    6.3
    7.1
    8.2
    8.4
    8.5
    9.1
   93.6
  100.0
```

and suppose that K-quantiles are available for K = 1, 4, 7, and 10:

```
  K   K-quantile
 ---  ----------
  1      0.0
  4      7.1
  7      8.5
 10    100.0
```

First consider the predicate `C <= 8.5`. For the data given above, exactly 7 rows satisfy this predicate. Assuming a uniform data distribution and using formula (1) from above, with KEY1 replaced by LOW2KEY, the number of rows that satisfy the predicate is estimated as:

```
  8.5 - 5.1
 ---------- x 10 *= 0
 93.6 - 5.1
```

where `*=` means “approximately equal to”. The error in this estimation is approximately -100%.

Using quantiles, the number of rows that satisfy this same predicate (`C <= 8.5`) is estimated by locating 8.5 as one of the K-quantile values and using the corresponding value of K, namely 7, as the estimate. In this case, the error is reduced to 0.

Now consider the predicate `C <= 10`. Exactly 8 rows satisfy this predicate. Unlike the previous example, the value 10 is not one of the stored K-quantiles. Assuming a uniform data distribution and using formula (1), the number of rows that satisfy the predicate is estimated as 1, an error of -86%.

Using quantiles, the optimizer estimates the number of rows that satisfy the predicate as `r_1 + r_2`, where `r_1` is the number of rows satisfying the predicate `C <= 8.5` and `r_2` is the number of rows satisfying the predicate `C > 8.5 AND C <= 10.`. As in the above example, `r_1 = 7`. To estimate `r_2` the optimizer uses linear interpolation:

```
        100.0 - 10.0
 r_2 *= ------------ x (# rows with value > 8.5 and <= 100.0)
        100.0 - 8.5
        100.0 - 10.0
      = ----------- x (10 - 7)
        100.0 - 8.5
     *= 3
```

The final estimate is `r_1 + r_2 *= 10`, and the absolute error is reduced by more than a factor of 3.

The reason that the use of quantiles improves the accuracy of the estimates in the above examples is that the real data values are ″clustered″ in the range 5 - 10, but the standard estimation formulas assume that the data values are spread out evenly between 0 and 100.

The use of quantiles also improves accuracy when there are significant differences in the frequencies of different data values. Consider a column having data values with the following frequencies:

Data Value	Frequency
20	5
30	5
40	15
50	50
60	15
70	5
80	5

Suppose that K-quantiles are available for K = 5, 25, 75, 95, and 100:

K	K-quantile
5	20
25	40
75	50
95	70
100	80

Also suppose that frequent value statistics are available based on the 3 most frequent values.

Consider the predicate `C BETWEEN 20 AND 30`. From the distribution of the data values, you can see that exactly 10 rows satisfy this predicate. Assuming a uniform data distribution and using formula (1), the number of rows that satisfy the predicate is estimated as:

```
30 - 20
-------  x 100 = 25
70 - 30
```

which has an error of 150%.

Using frequent-value statistics and quantiles, the number of rows that satisfy the predicate is estimated as `r_1 + r_2`, where `r_1` is the number of rows that satisfy the predicate (`C = 20`) and `r_2` is the number of rows that satisfy the predicate `C > 20 AND C <= 30`. Using formula (2), `r_1` is estimated as:

```
100 - 80
-------- = 5
  7 - 3
```

Using linear interpolation, `r_2` is estimated as:

```
  30 - 20
  ------- x (# rows with value > 20 and <= 40)
  40 - 20
```

```
    30 - 20
 = ------- x (25 - 5)
    40 - 20
 = 10,
```

yielding a final estimate of 15 and reducing the error by a factor of 3.

Collecting and Using Detailed Index Statistics

As an option, you may collect more detailed statistics on indexes that help the optimizer better estimate the cost of accessing a table using that index. This can be done in one of two ways: First, you can use the DETAILED clause on the RUNSTATS command; or, second, you can specify A, Y, or X for the `statsopt` parameter when calling the RUNSTATS API. The DETAILED statistics PAGE_FETCH_PAIRS and CLUSTERFACTOR will be collected only if the table is of a sufficient size: around 25 pages. In this case, CLUSTERFACTOR will be a value between 0 and 1; and CLUSTERRATIO will be -1 (not collected). For tables smaller than 25 pages, CLUSTERFACTOR will be -1 (not collected), and CLUSTERRATIO will be a value between 0 and 100; even if the DETAILED clause is specified for an index on that table.

Understanding Detailed Index Statistics

The DETAILED statistics attempt to capture, in a concise way, the number of physical I/Os that will be required to access the data pages of a table when a complete index scan is performed under different buffer sizes. As RUNSTATS scans through the pages of the index, it models the different buffer sizes, and gathers estimates of how often a page fault occurs. For example, with only 1 (one) buffer page available, every new page reference by the index will result in a page fault, and, in a worse case, every row could reference a different page, resulting in at most CARDINALITY I/Os. At the other extreme, when the buffer is big enough to hold the entire table (subject to the maximum buffer size), then each of the table’s NPAGES pages will be physically read exactly once. The number of physical I/Os must therefore be a monotone, non-increasing function of the buffer size.

RUNSTATS fits a piece-wise linear curve to these estimates, which is stored as a string of 11 pairs in the PAGE_FETCH_PAIRS statistic. The first value in each pair is a hypothetical buffer size, and the second value in each pair is the estimated number of physical I/Os to fetch the data pages in a complete scan of the index, with a buffer of that size totally available to that index scan. The optimizer then uses the PAGE_FETCH_PAIRS statistic to estimate the number of physical I/Os for data-page fetches in any complete or partial index scan using that index.

The shape of the curve stored in PAGE_FETCH_PAIRS for an index will depend upon the clustering behavior of that index.

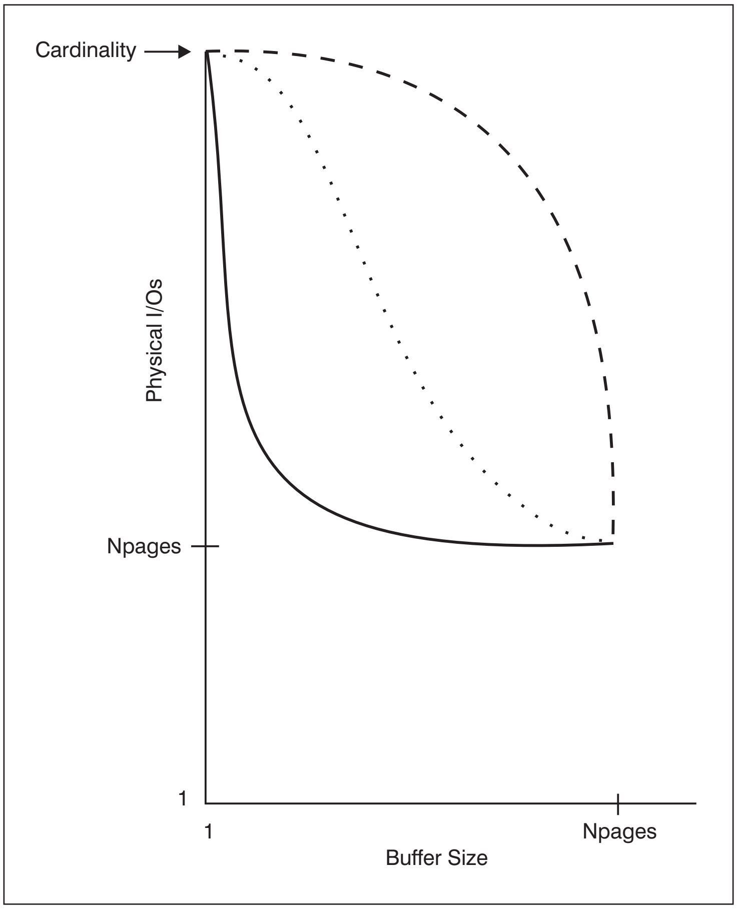


Figure 11. Three Curves for Clustered and Unclustered Indexes

There are three types of curves that are possible:

1. Curve 1 (dashed-line) is a highly-unclustered index that needs a buffer almost as large as the table before re-referenced pages are found in the buffer. This represents a situation in which references to the same page are widely spread throughout the index's key values, so a medium-sized buffer isn't sufficient to avoid re-referencing the same page multiple times. This is the worst scenario, as it requires the most buffer space to perform well. The optimizer is likely to use the list prefetch access strategy for such indexes, in an attempt to cluster the data-page accesses for the qualifying key values of the index. If this index is used frequently, it should be a prime candidate for reorganization.
2. Curve 2 (solid-line) is more locally unclustered. For very small buffers, it is as unclustered as curve 1, but once a few buffer pages are available to contain the most recently referenced data, the data-page hit ratio improves

significantly. This represents the somewhat favorable situation in which, although the index isn't particularly clustered, references to the same data pages are in a close proximity to one another among the index's key values.

3. Curve three (dotted-line) is somewhere between these two extremes, improving at a uniform rate as the buffer is increased. This is usually the more common case for unclustered indexes, and represents what the optimizer will assume in the absence of DETAILED indexes.

When Should You Use Detailed Index Statistics?

You should use DETAILED index statistics when your queries reference columns that are not all in the index. In addition, DETAILED index statistics should be used when:

- There are multiple unclustered indexes with varying degrees of clustering
- The degree of clustering is non-uniform among the key values
- The values in the index are updated non-uniformly.

It may be quite hard to determine these situations without previous knowledge, and without attempting to force an index scan under varying buffer sizes and using the monitor to observe the physical I/Os that result. Probably the cheapest way to determine whether any of these situations are occurring is to collect the DETAILED statistics for an index and retain them if the PAGE_FETCH_PAIRS that result are non-linear.

User Update-Capable Catalog Statistics

The ability to update selected system catalog statistics allows you to:

- Model query performance on a development system using production system statistics
- Perform "what if" query performance analysis.

You should **not** update statistics on a production system because you may hinder the optimizer from finding the best access plan for your query.

To update the values of these statistical columns, use the SQL UPDATE statement against the views defined in the SYSSTAT schema. You can update statistics for:

- Tables for which you hold explicit CONTROL privilege. You can also update statistics for columns and indexes for these tables.
- Nicknames for which you hold explicit CONTROL privilege in a federated database system. You can also update statistics for columns and indexes for these nicknames. Note that the update only affects local metadata (data source table statistics are not changed). These updates affect only the global access strategy generated by the DB2 optimizer.
- User-defined functions (UDFs) that you own (see "Updating Statistics for User-Defined Functions" on page 134 for guidance).

You can also update these statistics if your user ID has explicit DBADM authority for the database; that is, your user ID is recorded as having DBADM authority in the SYSCAT.DBAUTH table. Belonging to a DBADM group does not explicitly provide this authority.

Using these views, a DBADM can see statistics rows for all users. A user without DBADM authority can only see those rows which contain statistics for objects over which they have CONTROL privilege.

The following shows an example of updating the table statistics for the EMPLOYEE table:

```
UPDATE SYSSTAT.TABLES
SET  CARD   = 10000,
     NPAGES = 1000,
     FPAGES = 1000,
     OVERFLOW = 2
WHERE TABSCHEMA = 'userid'
  AND TABNAME   = 'EMPLOYEE'
```

You must be careful when updating catalog statistics. Arbitrary updates can have a serious impact on the performance of subsequent queries. You may wish to use any of the following methods to replace any updates you applied to these tables:
- ROLLBACK the unit of work in which the changes have been made (assuming the unit of work has not been committed).
- Using the RUNSTATS utility you can recalculate and refresh the catalog statistics.
- Update the catalog statistics to indicate that statistics have not been gathered. (For example, setting column NPAGES to -1 indicates that the number-of-pages statistic has not been collected.)
- Replace the catalog statistics with the data they contained prior to your update. This method would only be possible if you used the *db2look* tool, as described in “Modeling Production Databases” on page 135, to capture the statistics before you made any changes.

In a some cases, the optimizer may determine that some particular statistical value or combination of values are not valid, it will use default values and issue a warning. Such circumstances are rare, however, since most of the validation is done when updating the statistics.

Additional Information: For information about updating catalog statistics, see:
- “Rules for Updating Catalog Statistics” on page 130
- “Rules for Updating Table and Nickname Statistics” on page 130
- “Rules for Updating Column Statistics” on page 131
- “Rules for Updating Distribution Statistics for Columns” on page 131
- “Rules for Updating Index Statistics” on page 132

- “Updating Statistics for User-Defined Functions” on page 134
- “Modeling Production Databases” on page 135.

Rules for Updating Catalog Statistics

When you update catalog statistics, the most important general rule is to ensure that valid values, ranges, and formats of the various statistics are stored in the statistic views. It is also important to preserve the consistency of relationships between various statistics.

For example, COLCARD in SYSSTAT.COLUMNS must be less than CARD in SYSSTAT.TABLES (the number of distinct values in a column can’t be greater than the number of rows). Assume that you want to reduce COLCARD from 100 to 25, and CARD from 200 to 50. If you update SYSCAT.TABLES first, you should get an error (since CARD would be less than COLCARD). The correct order is to update COLCARD in SYSCAT.COLUMNS first, then update CARD in SYSSTAT.TABLES. The situation occurs in reverse if you want to increase COLCARD to 250 from 100, and CARD to 300 from 200. In this case, you must update CARD first, then COLCARD.

When a conflict is detected between an updated statistic and another statistic, an error is issued. However, errors may not always be issued when conflicts arise. In some situations, the conflict is difficult to detect and report in an error, especially if the two related statistics are in different catalogs. For this reason, you should be careful to avoid causing such conflicts.

The most common checks you should make, before updating a catalog statistic, are:

1. Numeric statistics must be -1 or greater than or equal to zero.
2. Numeric statistics representing percentages (for example, CLUSTERRATIO in SYSSTAT.INDEXES) must be between 0 and 100.

Note: For row types, the table level statistics NPAGES, FPAGES, and OVERFLOW are not updatable for a sub-table.

Rules for Updating Table and Nickname Statistics

There are only four statistic values that you can update in SYSTAT.TABLES: CARD, FPAGES, NPAGES, and OVERFLOW. Keep in mind that:

1. CARD must be greater than all COLCARD values in SYSSTAT.COLUMNS that correspond to that table.
2. CARD must be greater than NPAGES.
3. FPAGES must be greater than NPAGES.
4. NPAGES must be less than or equal to any ″Fetch″ value in the PAGE_FETCH_PAIRS column of any index (assuming this statistic is relevant for the index).

5. CARD must not be less than or equal to any ″Fetch″ value in the PAGE_FETCH_PAIRS column of any index (assuming this statistic is relevant for the index).

When working within a federated database system, use caution when manually providing/updating statistics on a nickname over a remote view. The statistical information, such as the number of rows this nickname will return, might not reflect the real cost to evaluate this remote view and thus might mislead the DB2 optimizer. Situations that can benefit from statistics updates include remote views defined on a single base table with no column functions applied on the SELECT list. Complex views may require a complex tuning process which might require that each query be tuned. Consider creating local views over nicknames instead so the DB2 optimizer knows how to derive the cost of the view more accurately.

Rules for Updating Column Statistics

When you are updating statistics in SYSSTAT.COLUMNS, follow the guidelines below. For details on updating column distribution statistics, see "Rules for Updating Distribution Statistics for Columns".

1. HIGH2KEY and LOW2KEY (in SYSSTAT.COLUMNS) must adhere to the following rules:
   - The datatype of any HIGH2KEY, LOW2KEY value must correspond to the datatype of the user column for which the statistic is attributed. Because HIGH2KEY is a VARCHAR column, you must enclose the value in quotation marks. For example, to set HIGH2KEY to 25 for an INTEGER user column, your update statement would include SET HIGH2KEY = '25'.
   - The length of HIGH2KEY, LOW2KEY values must be the smaller of 33 or the maximum length of the target column's datatype.
   - HIGH2KEY must be greater than LOW2KEY whenever there are 3 or more distinct values in the corresponding column. In the case of less than 3 distinct values in the column, HIGH2KEY can be equal to LOW2KEY.
2. The cardinality of a column (COLCARD statistic in SYSSTAT.COLUMNS) cannot be greater than the cardinality of its corresponding table (CARD statistic in SYSSTAT.TABLES).
3. The cardinality of a column (NUMNULLS statistic in SYSSTAT.COLUMNS) cannot be greater than the cardinality of its corresponding table (CARD statistic in SYSSTAT.TABLES).
4. No statistics are supported for columns with datatypes: LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB.

Rules for Updating Distribution Statistics for Columns

"User Update-Capable Catalog Statistics" on page 128 provides general information about how to update catalog statistics. You may wish to refer to that section before attempting to update column distribution statistics.

In order for all the statistics in the catalog to be consistent, you must exercise care when updating the distribution statistics. Specifically, for each column, the catalog entries for the frequent data statistics and quantiles must satisfy the following constraints:

1. Frequent value statistics (in the SYSSTAT.COLDIST catalog)
   - The values in column VALCOUNT must be non-increasing for increasing values of SEQNO.
   - The number of values in column COLVALUE must be less than or equal to the number of distinct values in the column, which is stored in column COLCARD in catalog view SYSSTAT.COLUMNS.
   - The sum of the values in column VALCOUNT must be less than or equal to the number of rows in the column, which is stored in column CARD in catalog view SYSSTAT.TABLES.
   - In most cases, the values in the column COLVALUE should lie between the second-highest and second-lowest data values for the column, which are stored in columns HIGH2KEY and LOW2KEY, respectively, in catalog view SYSSTAT.COLUMNS. There may be one frequent value greater than HIGH2KEY and one frequent value less than LOW2KEY.
2. Quantiles (in the SYSSTAT.COLDIST catalog)
   - The values in column COLVALUE must be non-decreasing for increasing values of SEQNO
   - The values in column VALCOUNT must be strictly increasing for increasing values of SEQNO
   - The largest value in column COLVALUE must have a corresponding entry in column VALCOUNT equal to the number of rows in the column
   - In most cases, the values in the column COLVALUE should lie between the second-highest and second-lowest data values for the column, which are stored in columns HIGH2KEY and LOW2KEY, respectively, in catalog view SYSSTAT.COLUMNS.

Suppose that distribution statistics are available for a column C1 with “R” rows and you wish to modify the statistics to correspond to a column with the same relative proportions of data values, but with “(F x R)” rows. To scale up the frequent-value statistics by a factor of F, each entry in column VALCOUNT must be multiplied by F. Similarly, to scale up the quantiles by a factor of F, each entry in column VALCOUNT must be multiplied by F. If these rules are not followed, the optimizer may use the wrong filter factor causing unpredictable performance when you run the query.

Rules for Updating Index Statistics

When you update the statistics in SYSSTAT.INDEXES, follow the rules described below:

1. PAGE_FETCH_PAIRS (in SYSSTAT. INDEXES) must adhere to the following rules:

- Individual values in the PAGE_FETCH_PAIRS statistic must be separated by a series of blank delimiters.
- Individual values in the PAGE_FETCH_PAIRS statistic must not be longer than 10 digits and must be less than the maximum integer value (MAXINT = 2147483647).
- There must always be a valid PAGE_FETCH_PAIRS value if the CLUSTERFACTOR is greater than zero.
- There must be exactly 11 pairs in a single PAGE_FETCH_PAIR statistic.
- Buffer size entries of PAGE_FETCH_PAIRS must be ascending in value.
- If the buffer size value is the same as that in the previous pair, the page fetch value must be the same as that in the previous pair.
- Any buffer size value in a PAGE_FETCH_PAIRS entry cannot be greater than MIN(NPAGES, 524287) where NPAGES is the number of pages in the corresponding table (in SYSSTAT.TABLES).
- "Fetches" entries of PAGE_FETCH_PAIRS must be descending in value, with no individual "Fetches" entry being less than NPAGES. "Fetch" size values in a PAGE_FETCH_PAIRS entry cannot be greater than the CARD (cardinality) statistic of the corresponding table.
- If buffer size value is the same in two consecutive pairs, then page fetch value must also be the same in both the pairs (in SYSSTAT.TABLES).

A valid PAGE_FETCH_UPDATE is:

```
PAGE_FETCH_PAIRS =
 '100 380 120 360 140 340 160 330 180 320 200 310 220 305 240 300
  260 300 280 300 300 300'
```

where

```
NPAGES = 300
CARD   = 10000
CLUSTERRATIO  =  -1
CLUSTERFACTOR = 0.9
```

2. CLUSTERRATIO and CLUSTERFACTOR (in SYSSTAT.INDEXES) must adhere to the following rules:
   - Valid values for CLUSTERRATIO are -1 or between 0 and 100.
   - Valid values for CLUSTERFACTOR are -1 or between 0 and 1.
   - At least one of the CLUSTERRATIO and CLUSTERFACTOR values must be -1 at all times.
   - If CLUSTERFACTOR is a positive value, it must be accompanied by a valid PAGE_FETCH_PAIR statistic.
3. The following rules apply to FIRSTKEYCARD, FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, and FULLKEYCARD:
   - FIRSTKEYCARD must be equal to FULLKEYCARD for a single-column index.
   - FIRSTKEYCARD must be equal to COLCARD for the corresponding column.

- If any of these index statistics are not relevant, you should set them to -1. For example, if you have an index with only 3 columns, set FIRST4KEYCARD to -1.
- For multiple column indexes, if all the statistics are relevant, the relationship between them must be:

```
FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
   <= FULLKEYCARD <= CARD
```

4. The following rules apply to SEQUENTIAL_PAGES and DENSITY:
   - Valid values for SEQUENTIAL_PAGES are -1 or between 0 and NLEAF.
   - Valid values for DENSITY are -1 or between 0 and 100.

Updating Statistics for User-Defined Functions

Using the SYSSTAT.FUNCTIONS catalog view, you may update statistics for user-defined functions (UDFs). If these statistics are available, the optimizer will use them when estimating costs for various access plans. If statistics are not available the statistic column values will be -1 and the optimizer will use default values that assume a simple UDF.

The following table provides information about the statistic columns that you may update for UDFs:

Table 13. Function Statistics (SYSCAT.FUNCTIONS and SYSSTAT.FUNCTIONS)

Statistic	Description
IOS_PER_INVOC	Estimated number of read/write requests executed each time a function is executed.
INSTS_PER_INVOC	Estimated number of machine instructions executed each time a function is executed.
IOS_PER_ARGBYTE	Estimated number of read/write requests executed per input argument byte.
INSTS_PER_ARGBYTES	Estimated number of machine instructions executed per input argument byte.
PERCENT_ARGBYTES	Estimated average percent of input argument bytes that the function will actually process.
INITIAL_IOS	Estimated number of read/write requests executed only the first/last time the function is invoked.
INITIAL_INSTS	Estimated number of machine instructions executed only the first/last time the function is invoked.
CARDINALITY	Estimated number of rows generated by a table function.

For example, consider a UDF (EU_SHOE) that converts an American shoe size to the equivalent European shoe size. (These two shoe sizes could be UDTs.) For this UDF, you should set the statistic columns as follows:

- INSTS_PER_INVOC should be set to the estimated number of machine instructions required to:
  - Invoke EU_SHOE
  - Initialize the output string
  - Return the result.
- INSTS_PER_ARGBYTE should be set to the estimated number of machine instructions required to convert the input string into a European shoe size.
- PERCENT_ARGBYTES would be set to 100 indicating that the entire input string is to be converted
- INITIAL_INSTS, IOS_PER_INVOC, IOS_PER_ARGBYTE, and INITIAL_IOS should all be set to 0, since this UDF only performs computations.

PERCENT_ARGBYTES would be used by a function that does not always process the entire input string. For example, consider a UDF (LOCATE) that accepts two arguments as input and returns the starting position of the first occurrence of the first argument within the second argument. Assume that the length of the first argument is small enough to be insignificant relative to the second argument and, on average, 75 percent of the second argument is searched. Based on this information, PERCENT_ARGBYTES should be set to 75. The above estimate of the average of 75 percent is based on the following additional assumptions:

- Half the time the first argument will not be found resulting in the entire second argument being searched
- The first argument is equally likely to appear anywhere within the second argument, resulting in half of the second argument being searched (on average) when the first argument is found.

INITIAL_INSTS or INITIAL_IOS can be used to record the estimated number of machine instructions or read/write requests performed only the first or last time the function is invoked. This could be used, for example, to record the cost of setting up a scratchpad area.

To obtain information about I/Os and instructions used by a user-defined function, you can use output provided by your programming language compiler or by monitoring tools available for your operating system.

Modeling Production Databases

Sometimes you may wish to have your test system contain a subset of your production system’s data. However, access plans selected for such a test system are not necessarily the same as those that would be selected on the production system, unless the catalog statistics and the configuration parameters for the test system are updated to match those of the production system.

A productivity tool, *db2look*, is provided that can be run against the production database to generate the update statements required to make the catalog statistics of the test database match those in production. These update statements can be generated by using *db2look* in mimic mode (`-m` option). In this case, *db2look* will generate a command processor script containing all the statements required to mimic the catalog statistics of the production database. This can be useful when analyzing SQL statements through Visual Explain in a test environment.

You can recreate database data objects, including tables, views, indexes, and other objects in a database, by extracting DDL statements with *db2look -e*. You can run the command processor script created from this command against another database to recreate the database. You can use the `-e` option with the `-m` option.

After running the update statements produced by `db2look` against the test system, the test system can be used to validate the access plans to be generated in production. Since the optimizer uses the type and configuration of the table spaces to estimate I/O costs, the test system must have the same table space geometry or layout. That is, the same number of containers of the same type: either SMS or DMS.

The *db2look* tool is found under the *bin* subdirectory.

For more information on how to use this productivity tool, type the following on a command line:

```
db2look -h
```

You can also refer to the *Command Reference* manual for more information on this tool.

The Control Center also provides an interface to the *db2look* utility called "Generate SQL - Object Name". Using the Control Center allows for the results file from the utility to be integrated into the Script Center. You can also schedule the *db2look* command from the Control Center. One difference when using the Control Center is that only single table analysis can be done as opposed to a maximum of thirty tables in a single call using the *db2look* command. You should also be aware that LaTex and Graphical outputs are not supported from the Control Center.

You can also run the `db2look` utility against an OS/390 database. The `db2look` utility extracts the DDL and UPDATE statistics statements for OS/390 objects. This is very useful if you would like to extract OS/390 objects and re-create them in a DB2 Universal Database (UDB) database. Refer to the *Command Reference* for additional information on the `db2look` utility.

There are some differences between the DB2 UDB statistics and the OS/390 statistics. The `db2look` utility performs the appropriate conversions from DB2 for OS/390 to DB2 UDB when this is applicable and sets to a default value (-1) the DB2 UDB statistics for which a DB2 for OS/390 counterpart does not exist. Here is how the `db2look` utility maps the DB2 for OS/390 statistics to DB2 UDB statistics. In the discussion below, "UDB_x" stands for a DB2 UDB statistics column; and, "S390_x" stands for a DB2 for OS/390 statistics column.

1. Table Level Statistics.

   UDB_CARD = S390_CARDF
   UDB_NPAGES = S390_NPAGES

   There is no S390_FPAGES. However, DB2 for OS/390 has another statistics called PCTPAGES which represents the percentage of active table space pages that contain rows of the table. So it is possible to calculate UDB_FPAGES based on S390_NPAGES and S390_PCTPAGES as follows:

   ```
   UDB_FPAGES=(S390_NPAGES * 100)/S390_PCTPAGES
   ```

   There is no S390_OVERFLOW to map to UDB_OVERFLOW. Therefore, the `db2look` utility just sets this to the default value:

   ```
   UDB_OVERFLOW=-1
   ```
2. Column Level Statistics.

   UDB_COLCARD = S390_COLCARDF
   UDB_HIGH2KEY = S390_HIGH2KEY
   UDB_LOW2KEY = S390_LOW2KEY

   There is no S390_AVGCOLLEN to map to UDB_AVGCOLLEN so the `db2look` utility just sets this to the default value:

   ```
   UDB_AVGCOLLEN=-1
   ```
3. Index Level Statistics.

   UDB_NLEAF = S390_NLEAF
   UDB_NLEVELS = S390_NLEVELS
   UDB_FIRSTKEYCARD= S390_FIRSTKEYCARD
   UDB_FULLKEYCARD = S390_FULLKEYCARD
   UDB_CLUSTERRATIO= S390_CLUSTERRATIO

   The other statistics for which there are no OS/390 counterparts are just set to the default. That is:

```
UDB_FIRST2KEYCARD = -1
UDB_FIRST3KEYCARD = -1
UDB_FIRST4KEYCARD  = -1
UDB_CLUSTERFACTOR = -1
UDB_SEQUENTIAL_PAGES = -1
UDB_DENSITY = -1
```

4. Column Distribution Statistics.

   There are two types of statistics in DB2 for OS/390 SYSIBM.SYSCOLUMNS. Type “F” for frequent values and type “C” for cardinality. Only entries of type “F” are applicable to DB2 for UDB and these are the ones that will be considered. Also, there is no column SEQNO in DB2 for OS/390 SYSIBM.SYSCOLUMNS but this is required for DB2 for UDB. Therefore, `db2look` generates one automatically.

   ```
   UDB_COLVALUE = S390_COLVALUE
   UDB_VALCOUNT = S390_FrequencyF * S390_CARD
   ```

Chapter 6. Understanding the SQL Compiler

When an SQL query is compiled, a number of steps are performed before the "best" access plan is either executed or stored in the system catalog.

In a partitioned database environment, all of the work done on a SQL query by the SQL Compiler takes place at the database partition to which you connect. Before being run, the compiled query is distributed to all database partitions in the database.

The following topics provide more information about the steps performed by the SQL Compiler:

- Overview of the SQL Compiler
- Rewrite Query by the SQL Compiler
- Operation Merging
- Operation Movement
- Predicate Translation
- Data Access Concepts and Optimization
- Optimization Strategies for Intra-Partition Parallelism
- Automatic Summary Tables
- Federated Database Query Compiler Phases

The following sections also provide information about factors external to the compiler which can affect the results produced by the compiler:

- "Chapter 3. Application Considerations" on page 41
- "Chapter 4. Environmental Considerations" on page 87
- "Chapter 5. System Catalog Statistics" on page 107.

"Chapter 7. SQL Explain Facility" on page 201 describes how you can examine the access plan chosen by the SQL compiler.

Overview of the SQL Compiler

The SQL compiler performs several steps before producing an access plan that you can execute. These steps are shown in Figure 12 on page 140.

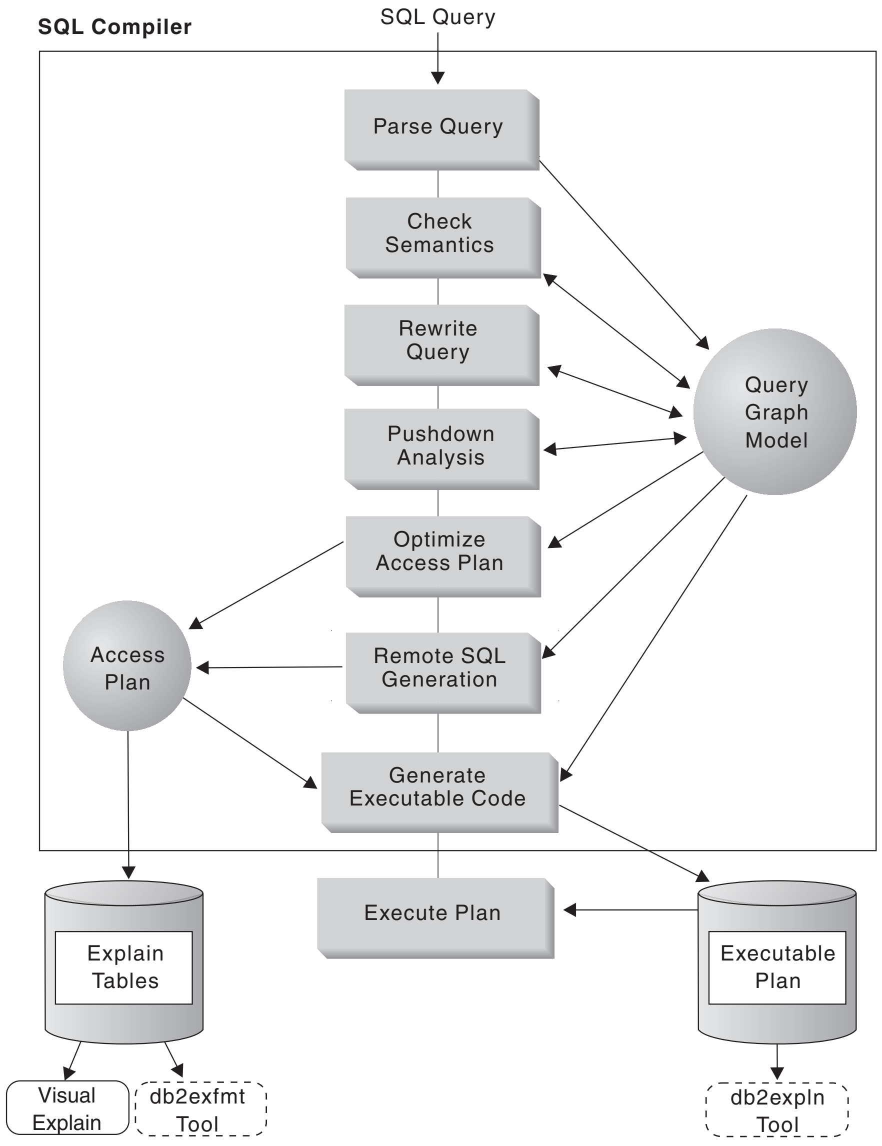


Figure 12. Steps performed by SQL Compiler

This diagram shows that the **Query Graph Model** is a key component of the SQL compiler. The *query graph model* is an internal, in-memory database that is used to represent the query throughout the query compilation process as described below:

- **Parse Query**

  The first task of the SQL compiler is to analyze the SQL query to validate the syntax. If any syntax errors are detected, the SQL compiler stops processing and the appropriate SQL error is returned to the application attempting to compile the SQL statement. When parsing is complete, an internal representation of the query is created.

- **Check Semantics**

  The second task of the compiler is to ensure there are no inconsistencies amongst parts of the statement. A simple example of this semantic checking is to ensure that the data type of the column specified for the YEAR scalar function is a datetime data type. Also during this second stage, the compiler adds the behavioral semantics to the query graph model, including the effects of referential constraints, table check constraints, triggers, and views.

  The query graph model contains all of the semantics of queries, including query blocks, subqueries, correlations, derived tables, expressions, data types, data type conversions, code page conversions, and partitioning keys.
- **Rewrite Query**

  The third phase of the SQL compiler uses the global semantics provided in the query graph model to transform the query into a form that can be optimized more easily. For example, the compiler might move a predicate, altering the level at which it is applied and potentially improving query performance. This type of operation movement is called *general predicate pushdown*. See “Rewrite Query by the SQL Compiler” on page 143 for more information.

  Working in a partitioned database environment, some query operations are more computationally intensive like those involving:
  - Aggregation
  - Redistribution of rows
  - Correlated subqueries.

  A *correlated subquery is a subquery that contains a reference to a column of a table that is outside the subquery.*

  In this environment, with some queries, decorrelation can occur as part of the rewrite of the query.

  The transferred query is stored in the Query Graph Model.
- **Pushdown Analysis (Federated Databases)**

  The major task of this step is to recommend to the DB2 optimizer whether an operation can be remotely evaluated (“pushed-down”) at a data source. This type of pushdown activity is specific to data source queries and represents an extension to general predicate pushdown operations.

  This step is bypassed unless you are executing federated database queries. See “Pushdown Analysis” on page 188 for more information.
- **Optimize Access Plan**

The SQL optimizer portion of the SQL compiler uses the query graph model as input, and generates many alternative execution plans for satisfying the user's request. It estimates the execution cost of each alternative plan, using the statistics for tables, indexes, columns and functions, and chooses the plan with the smallest estimated execution cost. The optimizer uses the query graph model to analyze the query semantics and to obtain information about a wide variety of factors, including indexes, base tables, derived tables, subqueries, correlations and recursion.

The optimizer portion can also consider a third type of pushdown operation: *aggregation and sort*, which can improve performance by pushing the evaluation of these operations to the Data Management Services component. See "Aggregation and Sort Pushdown Operators" on page 181 for more information.

The optimizer also considers whether there are different sized buffer pools when determining page size selection. That the environment includes a partitioned database is also considered as well as the ability to enhance the chosen plan for the possibility of intra-query parallelism in a symmetric multi-processor (SMP) environment. This information is used by the optimizer to help select the best access plan for the query. See "Data Access Concepts and Optimization" on page 153 for more information.

The output from this step of the SQL compiler is an "access plan". This access plan provides the basis for the information captured in the Explain tables. The information used to generate the access plan can be captured with an explain snapshot. (See "Chapter 7. SQL Explain Facility" on page 201 for more information on Explain topics.)

- **Remote SQL Generation (Federated Databases)**

  The final plan selected by the DB2 optimizer can consist of a set of steps that might operate on a remote data source. For those operations that will be performed by each data source, the remote SQL generation step creates an efficient SQL statement based on the data source SQL dialect.

  This step is bypassed unless you are executing federated database queries. See "Remote SQL Generation and Global Optimization" on page 195 for more information.

- **Generate "Executable" Code**

  The final step of the SQL Compiler uses the *access plan* and the query graph model to create an executable access plan, or section, for the query. This code generation step uses information from the query graph model to avoid repetitive execution of expressions that only need to be computed once for a query. Examples for which this optimization is possible include code page conversions and the use of host variables.

  Information about access plans for static SQL is stored in the system catalog tables. When the package is executed, the database manager will use the information stored in the system catalog tables to determine how to access the data and provide results for the query. It is this information that is used by the *db2expln* tool. (See “Chapter 7. SQL Explain Facility” on page 201 for more information on Explain topics.)

It is recommended that RUNSTATS be done periodically on tables used in queries where good performance is desired. The optimizer will then be better equipped with relevant statistical information on the nature of the data. If RUNSTATS is not done (or the optimizer suspects that RUNSTATS was done on empty or near empty tables), the optimizer may either use defaults or attempt to derive certain statistics based on the number of file pages used to store the table on disk (FPAGES).

Rewrite Query by the SQL Compiler

The SQL compiler includes a rewrite query stage which transforms SQL statements into forms that can be optimized more easily, and as a result, can improve the access path chosen. Rewriting queries is particularly important for queries which are very complex, including those queries with many subqueries or many joins. Query generator tools often create these types of very complex queries.

You can influence the number of query rewrite rules that are applied to an SQL statement by changing the optimization class (see “Adjusting the Optimization Class” on page 64).

You can see some of the results of the query rewrite through the use of the Explain facility or Visual Explain.

There are three major categories of rewriting that the SQL compiler may perform:
- Operation Merging
- Operation Movement
- Predicate Translation.

Operation Merging

The SQL compiler will rewrite queries to merge query operations, in an attempt to construct the query so that it has the fewest number of operations, especially SELECT operations. The following examples are provided to illustrate some of the operations that can be merged by the SQL compiler:
- Example - View Merges

  Using views in a SELECT statement can restrict the join order of the table and can also introduce redundant joining of tables. By merging the views during query rewrite, these restrictions can be lifted.
- Example - Subquery to Join Transformations

If the optimizer finds a subquery in a SELECT statement, it may be restriced in its selection of order processing of the tables.

- Example - Redundant Join Elimination

  During query rewrite redundant joins can be removed to further simplify the SELECT statement that will be optimized.
- Example - Shared Aggregation

  When using different functions, rewriting the query can reduce the number of calculations that need to be done.

Example - View Merges

Suppose you have access to the following two views of the EMPLOYEE table, one showing employees with a high level of education and the other view showing employees earning more than $35,000:

```
CREATE VIEW EMP_EDUCATION (EMPNO, FIRSTNME, LASTNAME, EDLEVEL) AS
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL
  FROM EMPLOYEE
 WHERE EDLEVEL > 17
CREATE VIEW EMP_SALARIES (EMPNO, FIRSTNAME, LASTNAME, SALARY) AS
SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
  FROM EMPLOYEE
 WHERE SALARY > 35000
```

Now suppose you perform the following query to list the employees who have a high education level and who are earning more than $35,000:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMP_EDUCATION E1,
       EMP_SALARIES  E2
 WHERE E1.EMPNO = E2.EMPNO
```

During query rewrite, these two views could be merged to create the following query:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

By merging the SELECT statements from the two views with the user-written SELECT statement, the optimizer can consider more choices when selecting an access plan. In addition, if the two views that have been merged use the same base table, additional rewriting may be performed as described in “Example - Redundant Join Elimination” on page 145.

Example - Subquery to Join Transformations

The SQL compiler will take a query containing a subquery, such as:

```
SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO
FROM EMPLOYEE
WHERE WORKDEPT IN
      (SELECT DEPTNO
         FROM DEPARTMENT
        WHERE DEPTNAME = 'OPERATIONS')
```

and convert it to a join query of the form:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME, PHONENO
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNAME = 'OPERATIONS'
```

A join is generally much more efficient to execute than a subquery.

Example - Redundant Join Elimination

Queries can sometimes be written or generated which have unnecessary joins. Queries such as the following could also be produced during the query rewrite stage as described in “Example - View Merges” on page 144.

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

In this query, the SQL compiler can eliminate the join and simplify the query to:

```
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL, SALARY
  FROM EMPLOYEE
 WHERE EDLEVEL > 17
   AND SALARY  > 35000
```

Another example assumes that a referential constraint exists between the EMPLOYEE and DEPARTMENT sample tables on the department number. First, a view is created.

```
CREATE VIEW PEPLVIEW
   AS SELECT FIRSTNME, LASTNAME, SALARY, DEPTNO, DEPTNAME, MGRNO
         FROM EMPLOYEE E DEPARTMENT D
         WHERE E.WORKDEPT = D.DEPTNO
```

Then a query such as the following:

```
SELECT LASTNAME, SALARY
   FROM PEPLVIEW
```

becomes

```
SELECT LASTNAME, SALARY
   FROM EMPLOYEE
   WHERE WORKDEPT NOT NULL
```

Note that in this situation, even if the user knows that the query can be re-written, they may not be able to do so because they do not have access to the underlying tables. They may only have access to the view (shown above). Therefore, this type of optimization has to be performed within the database manager.

Redundancy in referential integrity joins is likely where:

- Views are defined with joins
- Queries are automatically generated.

  For example, there are automated tools in query managers which prevent users from writing optimized queries.

Example - Shared Aggregation

Using multiple functions within a query can generate several calculations which take time. Reducing the number of calculations to be done within the query results in an improved plan. The SQL compiler takes a query using multiple functions such as:

```
SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  AVG(SALARY+BONUS+COMM) AS OAVG,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE;
```

and transforms the query in the following way:

```
SELECT OSUM,
  OSUM/OCOUNT
  OCOUNT
FROM (SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE) AS SHARED_AGG;
```

This rewrite reduces the query from 2 sums and 2 counts to 1 sum and 1 count.

Operation Movement

The SQL compiler will rewrite queries to move query operations in an attempt to construct the query with the minimum number of operations and predicates. The following examples are provided to illustrate some of the operations that can be moved by the SQL compiler:

- Example - DISTINCT Elimination

During query rewrite, the optimizer can move where the DISTINCT operation is performed, to reduce the cost of this operation. In the example provided, the DISTINCT operation is removed completely.

- Example - General Predicate Pushdown

  During query rewrite, the order of applying predicates can be changed so that more selective predicates are applied to the query as early as possible.
- Example - Decorrelation

  When in a partitioned database environment the movement of results sets between database partitions is costly. Reducing the size of what must be broadcast to other database partitions and/or the number of broadcasts is one of the objectives when rewriting queries.

Example - DISTINCT Elimination

If the EMPNO column was defined as the primary key of the EMPLOYEE table, the following query:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME
  FROM EMPLOYEE
```

would be rewritten by removing the DISTINCT clause:

```
SELECT EMPNO, FIRSTNME, LASTNAME
  FROM EMPLOYEE
```

In the above example, since the primary key is being selected, the SQL compiler knows that each row returned will already be unique. In this case, the DISTINCT key word is redundant. If the query was not rewritten, the optimizer would build a plan with the necessary processing (a sort, for example) to ensure that the columns are distinct.

Example - General Predicate Pushdown

Altering the level at which a predicate is normally applied can result in improved performance. For example, given the following view which provides a list of all employees in department “D11”:

```
CREATE VIEW D11_EMPLOYEE
 (EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM)
AS SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM
     FROM EMPLOYEE
    WHERE WORKDEPT = 'D11'
```

And given the following query:

```
SELECT FIRSTNME, PHONENO
  FROM D11_EMPLOYEE
 WHERE LASTNAME = 'BROWN'
```

The query rewrite stage of the compiler will push the predicate `LASTNAME = 'BROWN'` down into the view D11_EMPLOYEE. This allows the predicate to be applied sooner and potentially more efficiently. The actual query that could be executed in this example is:

```
SELECT FIRSTNME, PHONENO
  FROM EMPLOYEE
 WHERE LASTNAME = 'BROWN'
   AND WORKDEPT = 'D11'
```

Pushdown of predicates is not limited to views. Other situations in which predicates may be pushed down include UNIONs, GROUP BYs, and derived tables (nested table expressions or common table expressions).

Example - Decorrelation

In a partitioned database environment, the SQL compiler can rewrite the following query:

Find all the employees who are working on programming projects and are underpaid.

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
  FROM EMPLOYEE E, PROJECT P
 WHERE P.EMPNO = E.EMPNO
   AND P.PROJNAME LIKE '%PROGRAMMING%'
   AND E.SALARY+E.BONUS+E.COMM <
     (SELECT AVG(E1.SALARY+E1.BONUS+E1.COMM)
        FROM EMPLOYEE E1, PROJECT P1
        WHERE P1.PROJNAME LIKE '%PROGRAMMING%'
          AND P1.PROJNO = A.PROJNO
          AND E1.EMPNO = P1.EMPNO)
```

Since this query is correlated, and since both PROJECT and EMPLOYEE are unlikely to be partitioned on PROJNO, the broadcast of each project to each database partition is possible. In addition, the subquery would have to be evaluated many times.

The SQL compiler can rewrite the query as follows:

- Determine the distinct list of employees working on programming projects and call it DIST_PROJS. It must be distinct to ensure that aggregation is done once only for each project:

  ```
  WITH DIST_PROJS(PROJNO, EMPNO) AS
  (SELECT DISTINCT PROJNO, EMPNO
   FROM PROJECT P1
   WHERE P1.PROJNAME LIKE '%PROGRAMMING%')
  ```
- Using the distinct list of employees working on the programming projects, join this to the employee table, to get the average compensation per project, AVG_PER_PROJ:

```
AVG_PER_PROJ(PROJNO, AVG_COMP) AS
(SELECT P2.PROJNO, AVG(E1.SALARY+E1.BONUS+E1.COMM)
 FROM EMPLOYEE E1, DIST_PROJS P2
 WHERE E1.EMPNO = P2.EMPNO
 GROUP BY P2.PROJNO)
```

- Then the new query would be:

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
 FROM PROJECT P, EMPLOYEE E, AVG_PER_PROG A
WHERE P.EMPNO = E.EMPNO
  AND P.PROJNAME LIKE '%PROGRAMMING%'
  AND P.PROJNO = A.PROJNO
  AND E.SALARY+E.BONUS+E.COMM < A.AVG_COMP
```

The rewritten SQL query computes the `AVG_COMP` per project (`AVG_PRE_PROJ`) and can then broadcast the result to all database partitions containing the `EMPLOYEE` table.

Predicate Translation

The SQL compiler will rewrite queries to translate existing predicates to more optimal predicates for the specific query. The following examples are provided to illustrate some of the predicates that could be translated by the SQL compiler:

- Example - Addition of Implied Predicates

  During query rewrite, predicates can be added to the query to allow the optimizer to consider additional table joins when selecting the best access plan for the query.
- Example - OR to IN Transformations

  During query rewrite, an OR predicate can be translated into an IN predicate to allow for a more efficient access plan to be chosen. The SQL compiler can also translate an IN predicate into an OR predicate if this transformation would allow a more efficient access plan to be chosen.

Example - Addition of Implied Predicates

The following query produces a list of the managers whose departments report to “E01” and the projects for which those managers are responsible:

```
SELECT DEPT.DEPTNAME DEPT.MGRNO, EMP.LASTNAME, PROJ.PROJNAME
  FROM DEPARTMENT DEPT,
       EMPLOYEE   EMP,
       PROJECT    PROJ
 WHERE DEPT.ADMRDEPT = 'E01'
   AND DEPT.MGRNO = EMP.EMPNO
   AND EMP.EMPNO  = PROJ.RESPEMP
```

The query rewrite will add the following implied predicate:

```
DEPT.MGRNO = PROJ.RESPEMP
```

As a result of this rewrite, the optimizer can consider additional joins when it is trying to select the best access plan for the query.

In addition to the above predicate transitive closure, query rewrite will also derive additional local predicates based on the transitivity implied by equality predicates. For example, the following query lists the names of the departments (whose department number is greater than “E00”) and employees who work in that department.

```
SELECT EMPNO, LASTNAME, FIRSTNAME, DEPTNO, DEPTNAME
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNO > 'E00'
```

For this query, the rewrite stage will add the following implied predicate:

```
EMP.WORKDEPT > 'E00'
```

As a result of this rewrite, the optimizer reduces the number of rows to be joined.

Example - OR to IN Transformations

Suppose an OR clause connects two or more simple equality predicates on the same column, as in the following example:

```
SELECT *
  FROM EMPLOYEE
 WHERE DEPTNO = 'D11'
    OR DEPTNO = 'D21'
    OR DEPTNO = 'E21'
```

If there is no index on the DEPTNO column, converting the OR clause to the following IN predicate will allow the query to be processed more efficiently:

```
SELECT *
  FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

Note: In some cases, the database manager may convert an IN predicate to a set of OR clauses so that index ORing may be performed. See “Multiple Index Access” on page 159 for more information about index ORing.

Accounting for Column Correlation

You may have applications which contain queries constructed with joins that have more than one join predicate joining two tables. While this may sound complicated, such a situation is not unusual where you are attempting to determine relationships between similar, related columns between tables.

For example, a manufacturer makes products from raw material of various colors, elasticities and qualities. The finished product has the same color and elasticity as the raw material from which it is made. The manufacturer issues the query:

```
SELECT PRODUCT.NAME, RAWMATERIAL.QUALITY FROM PRODUCT, RAWMATERIAL
   WHERE PRODUCT.COLOR       =  RAWMATERIAL.COLOR
     AND PRODUCT.ELASTICITY  =  RAWMATERIAL.ELASTICITY
```

This query returns the names and raw material quality of all products. There are two join predicates:

```
PRODUCT.COLOR       =  RAWMATERIAL.COLOR
PRODUCT.ELASTICITY  =  RAWMATERIAL.ELASTICITY
```

When the DB2 UDB optimizer chooses a plan for executing this query, it calculates how selective each of the two predicates are, and assumes that they are independent, that is, that all variations of elasticity occur for each color, and that conversely for each level of elasticity there is raw material of every color. It then uses statistics on how many levels of elasticity and how many different colors there are in each table to calculate the overall selectivity of the pair of predicates. Based on this it may choose, for example, a Nested Loop Join in preference to a Merge Join, or vice versa.

However, it may be that these two predicates are not independent. For example, it may be that the highly elastic materials are available in only a few colors, and the very inelastic materials are only available in a few other colors (different from the elastic ones). Then the combined selectivity of the predicates is less (eliminates fewer rows) so the query will return more rows. To see this, imagine the extreme case where there is just one level of elasticity for each color and vice versa. Now either one of the predicates logically could be omitted entirely since it is implied by the other. The optimizer’s choice of plan may no longer be the best, for example it may be that the Nested Loop join plan is selected but the Merge Join would be faster.

With other database products, database administrators have tried to solve this performance problem by updating statistics in the catalog to try to make one of the predicates appear to be less selective, but this approach can cause unwanted side-effects on other queries.

DB2 UDB’s optimizer attempts to detect and compensate for correlation of join predicates if you:

1. Define unique indexes on the correlated columns, that is, on the columns of a table which appear in the correlated predicates.
2. Do not set the registry variable DB2_CORRELATED_PREDICATES to “NO”.

In the above example, you could define a unique index covering either:

```
   PRODUCT.COLOR, PRODUCT.ELASTICITY
```

or

```
   RAWMATERIAL.COLOR, RAWMATERIAL.ELASTICITY
```

or both.

In order for correlation to be detected, the non-include columns of this index must be correlated columns, and no other columns. The index may optionally contain include columns.

In general there may be more than 2 correlated columns in join predicates so you should ensure that you define the unique index to cover all of them.

In many cases the correlated columns in one table form its primary key. A primary key is always unique so if there's a primary key on the correlated columns, there's no need to define another unique index.

After doing this, ensure that statistics on tables are up to date and that they have not been altered away from the true values for any reason, for example to attempt to influence the optimizer.

The optimizer will use the FIRSTnKEYCARD and FULLKEYCARD information of the unique index statistics to detect cases of correlation, and dynamically adjust combined selectivities of the correlated predicates, thus obtaining a more accurate estimate of the join size and cost.

In addition to JOIN predicate correlation, the optimizer also accounts for correlation with simple equal predicates of the type COL = "constant". For example, consider a table of different types of cars, each having a MAKE (that is, a manufacturer), MODEL, STYLE (that is, sedan, station wagon, sports utility vehicle), YEAR, and COLOR. Predicates on COLOR are likely to be independent of those on MAKE, MODEL, STYLE, or YEAR, since almost every manufacturer makes the same standard colors available on each of their models and styles, year after year. However, the predicates MAKE and MODEL certainly are not independent since only a single car maker would make a model with a particular name. Identical model names used by two or more car makers is very unlikely and certainly not wanted by the car makers. If an index exists on the two columns MAKE and MODEL, the optimizer will use the statistics from the index to determine the combined number of distinct values and adjust the selectivity or cardinality estimation for correlation between the two columns. For such predicates which are not join predicates, one need not have a unique index for the optimizer to make the adjustment.

Data Access Concepts and Optimization

When compiling an SQL statement, the SQL optimizer estimates the execution cost of different ways of satisfying your request. Based on this evaluation, the optimizer selects what it believes to be the optimal access plan. An *access plan* specifies the order of operations required to resolve an SQL statement. When an application program is bound, a *package* is created. This package contains access plans for all of the static SQL statements in that application program. Access plans for dynamic SQL statements are created at the time that the application is executed.

There are two ways of accessing data in a table: by directly reading the table (relation scan), or by first accessing an index on that table (index scan).

A *relation scan* occurs when the database manager sequentially accesses every row of a table. See “Index Scan Concepts” to learn how an index scan works and see “Relation Scan versus Index Scan” on page 162 to understand under what conditions each type of scan is used.

The following topics describe other methods that can also be used in an access plan to access data in a table, and to produce the results for your query:

- “Predicate Terminology” on page 162
- “Join Concepts” on page 165
- “Join Strategies in a Partitioned Database” on page 173
- “Influence of Sorting on the Optimizer” on page 180.

Other Related Topics:

- “Adjusting the Optimization Class” on page 64, provides information about controlling the number of alternative access plans evaluated by the SQL compiler
- “Chapter 7. SQL Explain Facility” on page 201, provides information about how you can obtain information about the access plan chosen by the SQL compiler.

Index Scan Concepts

An *index scan* occurs when the database manager accesses an index to do any or all of the following:

- Narrow down the set of qualifying rows (by scanning the rows in a certain range of the index) before accessing the base table. The index *scan range* (the start and stop points of the scan) is determined by the values in the query against which index columns are being compared.
- Order the output.
- Fully retrieve the requested data. If all of the requested data is in the index, the base table will not be accessed. This is known as an *Index-only access*.

Scans may also be performed on indexes in the direction opposite to that with which they were defined. Refer to the ALLOW REVERSE SCANS option on the CREATE INDEX statement in the *SQL Reference* for more information.

The following additional topics are provided:

- Index Structure
- Index Scans to Delimit a Range
- Index Scans to Order Data
- Index-Only Access
- Multiple Index Access
- Clustered Indexes
- Index Page Prefetch.

Index Structure

The database manager uses a B+ tree structure for storing its indexes. A B+ tree has one or more levels, as shown in the following diagram (where `RID` means row ID):

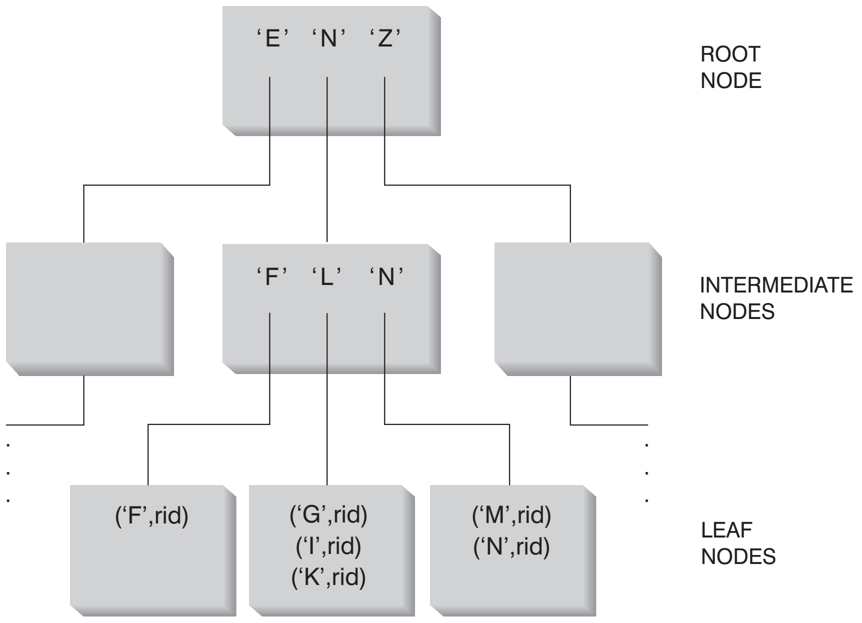


Figure 13. B+ Tree Structure

The top level is called the *root node*. The bottom level consists of *leaf nodes*, where the actual index key values are stored, as well as a pointer to the actual row in the table. Levels between the root and leaf node levels are called *intermediate nodes*.

In looking for a particular index key value, Index Manager searches the index tree, starting at the root node. The root contains one key for each node at the next level. The value of each of these keys is the largest existing key value for the corresponding node at the next level. For example, if an index has three levels as shown in Figure 13 on page 154, then to find an index key value, Index Manager would search the root node for the first key value greater than or equal to the key being looked for. This root node key would point to a specific intermediate node. The same procedure would be followed with that intermediate node to determine which leaf node to go to. The final index key would be found in the leaf node. Using Figure 13 on page 154, the key being looked for is "I". The first key in the root node greater than or equal to "I" is "N". This points to the middle node at the next level. The first key in that intermediate node that is greater than or equal to "I" is "L". This points to a specific leaf node where the index key for "I" along with its corresponding row ID(s) are found (the row ID of the corresponding rows in the base table).

Note: At the leaf node level there can be previous leaf pointers. This can be of great benefit since once finding a particular key value in the index by traversing the tree, the Index Manager can scan through the leaf nodes in either direction to retrieve a range of values. This ability to scan in either direction is only possible if the index was created using the ALLOW REVERSE SCANS parameter.

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

Index Scans to Delimit a Range

In determining whether an index can be used for a particular query, the optimizer evaluates each column of the index starting with the first column to see if it can be used to satisfy:

- Any of the EQUAL predicates in the statement's WHERE clause
- Any other predicates in the WHERE clause.

A *predicate* is an element of a search condition in a WHERE clause that expresses or implies a comparison operation. Predicates that can be used to delimit the range of an index scan are those involving an index column in which one of the following is true:

- The index column is being tested for equality against a constant, a host variable, an expression that evaluates to a constant, or a keyword
- The test against the index column is "IS NULL" or "IS NOT NULL"
- The test is for equality against a basic subquery (that is, one that does not contain ANY, ALL, or SOME), and the subquery does not have a correlated column reference to its immediate parent query block (that is, the SELECT for which this subquery is a subselect).
- The test is an inequality predicate meeting the conditions described below.

For example, given an index with the following definition:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

the following predicates could be used in delimiting the range of the scan of index IX1:

```
WHERE  NAME = :hv1
  AND  DEPT = :hv2
```

or

```
WHERE  MGR  = :hv1
  AND  NAME = :hv2
  AND  DEPT = :hv3
```

Note that in the second example the WHERE predicates do not have to be specified in the same order as the key columns appear in the index. And, although host variables are used in the examples, parameter markers, expressions, or constants would have the same effect.

A single index created using the ALLOW REVERSE SCANS parameter on the CREATE INDEX statement can be scanned in a forward or a backward direction. That is, such indexes support scans in the direction defined when the index was created and scans in the opposite or reverse direction. The statement could look something like this:

```
CREATE INDEX iname ON tname (cname DESC) ALLOW REVERSE SCANS
```

In this case, the index (`iname`) is formed based on DESCending values in `cname`. By allowing reverse scans, although the index on the column is defined for scans in descending order, a scan can be done in ascending order. The actual use of the index in both directions is not controlled by you but by the optimizer when creating and considering access plans.

In the following WHERE clause, only the predicates for NAME and DEPT would be used in delimiting the range of the index scan, but not the predicates for SALARY or YEARS:

```
WHERE  NAME   = :hv1
  AND  DEPT   = :hv2
  AND  SALARY = :hv4
  AND  YEARS  = :hv5
```

This is because there is a key column (MGR) separating these columns from the first two index key columns, so the ordering would be off. However, once the range is determined by the NAME = :hv1 and DEPT = :hv2 predicates, the remaining predicates can be evaluated against the remaining index key columns.

In addition to the equality predicates described above, certain inequality predicates may be used to delimit the range of an index scan. The following discusses the two types of inequality predicates: strict inequality and inclusive inequality.

Strict Inequality Predicates: The strict inequality operators which can be used for range delimiting predicates are > and <.

For delimiting a range for an index scan, only one column with strict inequality predicates will be considered. In the following example, the predicates on the NAME and DEPT columns can be used to delimit the range, but the predicate on the MGR column cannot be used.

```
WHERE  NAME   = :hv1
  AND  DEPT   > :hv2
  AND  DEPT   < :hv3
  AND  MGR    < :hv4
```

Inclusive Inequality Predicates: The following are inclusive inequality operators which can be used for range delimiting predicates:

- >= and <=
- BETWEEN
- LIKE

For delimiting a range for an index scan, multiple columns with inclusive inequality predicates will be considered. In the following example, all of the predicates can be used to delimit the range of the index scan:

```
WHERE  NAME   = :hv1
  AND  DEPT  >= :hv2
  AND  DEPT  <= :hv3
  AND  MGR   <= :hv4
```

To further illustrate this example, suppose that `:hv2 = 404`, `:hv3 = 406`, and `:hv4 = 12345`. The database manager will scan the index for all of departments 404 and 405, but it will stop scanning department 406 when it reaches the first manager that has an employee number (MGR column) greater than 12345.

For additional information, see “Range Delimiting and Index SARGable Predicates” on page 163.

Index Scans to Order Data

If the query involves ordering, an index can be used to order the data if the ordering columns appear consecutively in the index, starting from the first index key column. (Ordering or sorting can result from operations such as ORDER BY, DISTINCT, GROUP BY, "= ANY" subquery, "> ALL" subquery, "< ALL" subquery, INTERSECT or EXCEPT, UNION.) An exception to this is when the index key columns are compared for equality against "constant values" (that is, any expression that evaluates to a constant). In this case the ordering column can be other than the first index key columns. For example, in the query:

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY MGR
```

the index could be used to order the rows since NAME and DEPT will always be the same values and will thus be ordered. Another way of saying this is that the preceding WHERE and ORDER BY clauses are equivalent to:

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY NAME, DEPT, MGR
```

A unique index can also be used to truncate an order requirement. For example, given the following index definition and order by clause:

```
UNIQUE INDEX IX0:  PROJNO  ASC
SELECT PROJNO, PROJNAME, DEPTNO
  FROM PROJECT
ORDER BY PROJNO, PROJNAME
```

additional ordering on the PROJNAME column is not required since the IX0 index ensures that PROJNO is unique. This uniqueness ensures that there is only one PROJNAME value for each PROJNO value.

Index-Only Access

In some cases, all of the required data can be retrieved from the index without accessing the table. This is known as an *index-only* access.

To illustrate an index-only access, consider the following index definition:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

and the following query can be satisfied by accessing only the index, and without reading the base table:

```
SELECT NAME, DEPT, MGR, SALARY
  FROM EMPLOYEE
 WHERE NAME = 'SMITH'
```

In other cases, there may be columns that do not appear in the index. To obtain the data for these columns, rows of the base table must be read. For example, given the IX1 index, the following query needs to access the base table to obtain the PHONENO and HIREDATE column data:

```
SELECT NAME, DEPT, MGR, SALARY, PHONENO, HIREDATE
  FROM EMPLOYEE
  WHERE NAME = 'SMITH'
```

By creating a unique index with include columns, you can improve the performance of data retrieval by increasing the number of access attempts based solely on indexes.

To illustrate the use of include columns, consider the following index definition:

```
CREATE UNIQUE INDEX IX1 ON EMPLOYEE
   (NAME ASC)
    INCLUDE (DEPT, MGR, SALARY, YEARS)
```

This creates a unique index which enforces uniqueness of the NAME column yet stores and maintains data for DEPT, MGR, SALARY, and YEARS columns.

The following query can be satisfied by accessing only the index and without reading the base table:

```
SELECT NAME, DEPT, MGR, SALARY
   FROM EMPLOYEE
   WHERE NAME='SMITH'
```

Multiple Index Access

In all of the above examples, a single index scan was performed to produce the results. To satisfy the predicates of a WHERE clause, the optimizer can choose to scan multiple indexes. For example, given the following two index definitions:

```
INDEX IX2:  DEPT    ASC
INDEX IX3:  JOB     ASC,
            YEARS   ASC
```

the following predicates could be resolved using these two indexes:

```
WHERE DEPT = :hv1
  OR (JOB   = :hv2
  AND YEARS >= :hv3)
```

In this example, scanning index IX2 will produce a list of row IDs (RIDs) that satisfy the `DEPT = :hv1` predicate. Scanning index IX3 will produce a list of RIDs satisfying the `JOB = :hv2 AND YEARS >= :hv3` predicate. These two lists of RIDs can be combined and duplicates removed before accessing the table. This is known as *index ORing*.

Index ORing may also be used for predicates using the IN expression, as in the following example:

```
WHERE DEPT IN (:hv1, :hv2, :hv3)
```

The objective of index ORing is to eliminate duplicate RIDs; however, the objective of *index ANDing* is to find common RIDs. Index ANDing may occur with applications where there are multiple indexes on corresponding columns within the same table and a query using multiple "and" predicates is run against that table. Multiple index scans against each indexed column in such a query produce values which are hashed to create bitmaps. The second bitmap is used to probe the first bitmap to generate the qualifying rows that are fetched to create the final returned data set.

For example, given the following two index definitions:

```
INDEX IX4: SALARY   ASC
INDEX IX5: COMM     ASC
```

the following predicates could be resolved using these two indexes:

```
WHERE SALARY BETWEEN 20000 AND 30000
   AND COMM BETWEEN 1000 AND 3000
```

In this example, scanning index `IX4` produces a bitmap satisfying the `SALARY BETWEEN 20000 AND 30000` predicate. Scanning `IX5` and probing the bitmap for `IX4` results in the list of qualifying RIDs that satisfy both predicates. This is known as "dynamic bitmap ANDing". It occurs only if the table has sufficient cardinality and the columns have sufficient values in the qualifying range, or sufficient duplication if equality predicates are used.

Note: In the accessing of any single table, DB2 does not combine index ANDing and index ORing.

Clustered Indexes

When selecting the access plan, the optimizer considers the I/O cost of fetching pages from disk to the buffer pool. In its calculations, the optimizer will estimate the number of I/Os required to satisfy a query. This estimate includes a prediction of buffer pool usage, since additional I/Os are not required to read rows in a page that is already in the buffer pool.

For index scans, the optimizer uses information from the system catalog tables (SYSCAT.INDEXES) to help estimate I/O cost of reading data pages into the buffer pool. The following columns from the SYSCAT.INDEXES table are used:

- CLUSTERRATIO indicating the degree to which the table data in relation to this index is clustered. A higher number means that the rows are ordered on the data pages in index key sequence. Therefore, all of the rows on a data page can be read while the page is in buffer. If the value of this column is -1, the optimizer will attempt to use PAGE_FETCH_PAIRS and CLUSTERFACTOR.

or

- PAGE_FETCH_PAIRS containing several pairs of numbers which model the number of I/Os required to read the data pages into buffer pools of various sizes together with CLUSTERFACTOR. When collecting statistics for an index, this information is considered a detailed statistic.

If statistics are not available, the optimizer will use default values for the statistics, which assume poor clustering of the data to the index. See also "Chapter 5. System Catalog Statistics" on page 107 and "Collecting Statistics Using the RUNSTATS Utility" on page 108.

You can specify a clustering index that will be used both to cluster the rows during a table reorganization and to preserve this characteristic during insert processing. (See "Reorganizing Catalogs and User Tables" on page 251 for information about table reorganization.) Subsequent updates and inserts may make the index less well clustered (as measured by the statistics gathered by RUNSTATS), so you may need to periodically reorganize the table. To reduce the frequency of reorganization on a volatile database, use the PCTFREE parameter when altering a table. This will allow for additional inserts to be clustered with the existing data.

The degree to which the data is clustered with respect to the index can have a significant impact on performance and you should try to keep one of the indexes on the table close to 100 percent clustered.

In general, only one index can be one hundred percent clustered, except in those cases where the keys are a superset of the keys of the clustering index; or, where there is de facto correlation between the key columns of the two indexes.

See "Performance Tips for Administering Indexes" on page 98 for more information on performance reasons to use clustering indexes. Refer to the *SQL Reference*, CREATE INDEX, for more information on how to create a clustering index.

Clustering Page Reads Using List Prefetch: If the optimizer uses an index to access rows, it can defer reading the data pages until all the RIDs (row identifiers) have been obtained from the index. For example, given the previously defined index IX1:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

and the following search criteria:

```
WHERE NAME BETWEEN 'A' and 'I'
```

the optimizer could perform an index scan on IX1 to determine the rows (and data pages) to retrieve. If the data was not clustered according to this index, list prefetch will include a step to sort the list of RIDs obtained from the index scan. See "Understanding List Prefetching" on page 243 for more information.

Index Page Prefetch

When appropriate, the database manager detects sequential access to index pages and will generate prefetch requests. This will significantly reduce the elapsed time for nonselective index scans, and selective index scans accessing a significant portion of the index.

The optimizer uses index statistics such as DENSITY and SEQUENTIAL_PAGES, the characteristics of the table spaces in which the index resides, and the effect of any range delimiting predicates, to estimate the amount of index page prefetch that will occur. These estimates are factored into the overall cost estimate for using a particular index.

See "Understanding Sequential Prefetching" on page 242 for more information.

Relation Scan versus Index Scan

The optimizer will choose a relation scan when an index cannot be used for the query, or if the optimizer determines that an index scan would be more costly. An index scan could be more costly when:

- The table is small
- Index clustering is low
- Most of the table is accessed.

You may use the SQL Explain facilities to determine whether your access plan uses a relation scan or an index scan. See "Chapter 7. SQL Explain Facility" on page 201.

Predicate Terminology

A user application requests a set of rows from the database with an SQL statement, qualifying the specific rows desired through the use of predicates. When the optimizer decides how to evaluate an SQL statement, each predicate falls into one of four categories. The category is determined by how and when that predicate is used in the evaluation process. These categories are listed below, ordered in terms of performance from best to worst:

1. Range delimiting predicates
2. Index SARGable predicates
3. Data SARGable predicates
4. Residual predicates.

SARGable refers to something that can be used as a *s*earch *arg*ument.

"Summary of Predicate Usage" on page 164 provides a comparison of the characteristics that affect the performance of the various predicate categories.

Range Delimiting and Index SARGable Predicates

Range delimiting predicates are those used to bracket an index scan. They provide start and/or stop key values for the index search. Index SARGable predicates are not used to bracket a search, but can be evaluated from the index because the columns involved in the predicate are part of the index key. For example, given the previously defined index IX1 (in the section "Index Scan Concepts" on page 153) and the following WHERE clause:

```
WHERE  NAME  = :hv1
 AND   DEPT  = :hv2
 AND   YEARS > :hv5
```

the first two predicates (NAME = :hv1, DEPT = :hv2) would be range delimiting predicates, while YEARS > :hv5 would be an index SARGable predicate.

The database manager will make use of the index data in evaluating these predicates rather than reading the base table. These *index SARGable* predicates reduce the number of data pages accessed by reducing the set of rows that need to be read from the table. These types of predicates do not affect the number of index pages that are accessed.

Data SARGable Predicates

Predicates that cannot be evaluated by Index Manager, but can be evaluated by Data Management Services are called *data SARGable* predicates. Typically, these predicates require the access of individual rows from a base table. If required, Data Management Services will retrieve the columns needed to evaluate the predicate, as well as any others to satisfy the columns in the SELECT list that could not be obtained from the index.

For example, given a single index defined on the PROJECT table:

```
INDEX IX0:  PROJNO ASC
```

And given the following query, the DEPTNO = 'D11' predicate is considered to be data SARGable.

```
    SELECT PROJNO, PROJNAME, RESPEMP
      FROM PROJECT
     WHERE DEPTNO = 'D11'
     ORDER BY PROJNO
```

Residual Predicates

Residual predicates, typically, are those that require I/O beyond the simple accessing of a base table. Examples of residual predicates include those using correlated subqueries, using quantified subqueries (subqueries with ANY, ALL, SOME, or IN), or reading LONG VARCHAR or LOB data (stored in a file separate from the table). These predicates are evaluated by Relational Data Services.

Sometimes predicates, which are applied to the index only, have to be reapplied when the data page is accessed. For example, access plans using index ORing or index ANDing, (see “Multiple Index Access” on page 159), always reapply the predicates as residual predicates, when the data page is accessed.

Summary of Predicate Usage

The use of predicates in a query can help to reduce the amount of data read to satisfy the query. Different categories of predicates have different impacts on the performance of a query and these impacts are considered by the optimizer. The following table shows the ranking of the different types of predicates and how each type of predicate can influence performance.

Table 14. Summary of Predicate Type Characteristics

Characteristic	**Predicate Type**			
	Range Delimiting	**Index SARGable**	**Data SARGable**	**Residual**
Reduce index I/O	Yes	No	No	No
Reduce data page I/O	Yes	Yes	No	No
Reduce number of rows passed internally	Yes	Yes	Yes	No
Reduce number of qualifying rows	Yes	Yes	Yes	Yes

Join Concepts

A *join* is where rows from one table are concatenated to rows of one or more other tables. For example, given the following two tables:

```
        TABLE1                     TABLE2
    -----------------          -----------------
     PROJ     PROJ_ID          PROJ_ID     NAME
    ------    -------          -------    ------
      A          1                1        Sam
      B          2                3        Joe
      C          3                4        Mary
      D          4                1        Sue
                                  2        Mike
```

Joining Table1 and Table2 where the ID columns are equal would be represented by the following SQL statement:

```
   SELECT PROJ, x.PROJ_ID, NAME
     FROM TABLE1 x, TABLE2 y
     WHERE x.PROJ_ID = y.PROJ_ID
```

and would yield the following set of result rows:

```
     PROJ     PROJ_ID     NAME
    ------    -------    ------
      A          1        Sam
      A          1        Sue
      B          2        Mike
      C          3        Joe
      D          4        Mary
```

When joining two tables, one table is selected as the outer table and the other as the inner. The outer table is accessed first and is only scanned once. Whether the inner table is scanned multiple times depends on the type of join and which indexes are present. Whether your query joins two tables or more than two tables, the optimizer will only join two tables at a time. If needed, temporary, intermediary results tables will be created.

The optimizer will choose one of the two join methods (nested loop join or merge join) depending on the existence of a join predicate (defined in "Merge Join" on page 167), as well as various costs involved as determined by table and index statistics.

Nested Loop Join

A nested loop join is performed in one of two ways:

1. By scanning through the inner table for each accessed row of the outer table

   For example, if column A in tables T1 and T2 has the following values:

   ```
   Outer Table T1: column A      Inner Table T2: column A
   ------------------------      ------------------------
             2                             3
   ```

```
            3                            2
            3                            2
                                         3
                                         1
```

The steps for doing the nested loop:

- Read the first row from T1. The value for A is “2”
- Scan T2 until a match (“2”) is found, and then join the two rows
- Scan T2 until the next match (“2”) is found, and then join the two rows
- Scan T2 to the end of the table
- Go back to T1 and read the next row (“3”)
- Scan T2, starting at the first row, until a match (“3”) is found, and then join the two rows
- Scan T2 until the next match (“3”) is found, and then join the two rows
- Scan T2 to the end of the table
- Go back to T1 and read the next row (“3”)
- Scan T2 as before, joining all rows which match (“3”).

2. By doing an index lookup on the inner table for each accessed row of the outer table.

   This method can be used for the specified predicates if there is a predicate of the following form:

   ```
   expr(outer_table.column)  relop  inner_table.column
   ```

   where `relop` is a relative operator (for example =, >, >=, <, or <=) and `expr` is a valid expression on the outer table. The following are examples:

   ```
   OUTER.C1 + OUTER.C2 <= INNER.C1
   ```

   and

   ```
   OUTER.C4 < INNER.C3
   ```

   This method could be a way to significantly reduce the number of rows accessed in the inner table for each access of the outer table (although it depends on a number of factors, including the selectivity of the join predicate).

When evaluating a nested loop join, the optimizer will also determine whether or not to sort the outer table before performing the join. By ordering the outer table, based on the join columns, the number of read operations to access pages from disk for the inner table may be reduced, since it is more likely they will already be in the buffer pool. If the join uses a highly clustered index to access the inner table, the number of index pages accessed may be minimized if the outer table has been sorted.

In addition, the optimizer may also choose to perform the sort before the join, if it expects that the join will make a later sort more expensive. A later sort could be required to support a GROUP BY, DISTINCT, ORDER BY or merge join.

Merge Join

Merge join (sometimes known as merge scan join or sort merge join) requires a predicate of the form table1.column = table2.column. This is called an *equality join predicate*. Merge join requires ordered input on the joining columns, either through index access or by sorting. In order for a merge join to be used, the join column cannot be a LONG field column or a large object (LOB) column.

The joined tables are scanned simultaneously. The outer table of the merge join is scanned just once. The inner table is also scanned once unless there are repeated values in the outer table. If there are repeated values in the outer table, a group of rows in the inner table may be scanned again. For example, if column A in tables T1 and T2 has the following values:

```
    Outer Table T1: column A      Inner Table T2: column A
    ------------------------      ------------------------
             2                              1
             3                              2
             3                              2
                                            3
                                            3
```

The steps for doing the merge join are:
- Read the first row from T1. The value for A is “2”
- Scan T2 until a match is found, and then join the two rows
- Keep scanning T2 while the columns match, joining rows.
- When the “3” in T2 is read, go back to T1 and read the next row
- The next value in T1 is “3”, which matches T2, so join the rows
- Keep scanning T2 while the columns match, joining rows
- The end of T2 is reached
- Go back to T1 to get the next row — note that the next value in T1 is the same as the previous value from T1, so T2 is scanned again starting at the first “3” in T2 (the database manager remembers this position).

Hash Join

Hash join requires one or more predicates of the form table1.columnX = table2.columnY, and for which the column types are the **same**. For columns of type CHAR, the length must be the same. For columns of type DECIMAL, the precision and scale must be the same. The column type cannot be a LONG field column, or a large object (LOB) column.

First, one table (called the INNER table) is scanned and the rows copied into memory buffers drawn from the sort heap allocation (see the "Sort Heap Size (sortheap)" on page 342 database configuration parameter). The memory buffers are divided into partitions based on a "hash code" computed from the column(s) of the join predicate(s). If the size of the first table exceeds the available sort heap space, buffers from selected partitions are written to temporary tables. After finishing the processing of the INNER table, the second table (called the OUTER table) is scanned. Rows of the OUTER table are matched to rows from the INNER table by first comparing a "hash code" generated from the columns of the join predicate(s). Then, if the "hash code" of the OUTER row matches the "hash code" of the INNER row, the actual join predicate columns are compared.

OUTER table rows corresponding to partitions not written to a temporary table are matched immediately with INNER table rows in memory. Otherwise, if the corresponding INNER table partition was written to a temporary table, the OUTER row is also written to a temporary table. Finally, matching pairs of partitions from temporary tables are read and the "hash codes" of their rows are matched and join predicates checked.

To realize the performance benefits of hash join, it may be necessary to change the value of the *sortheap* database configuration parameter, and the *sheapthres* database manager configuration parameter.

For decision support queries, hash join access plans use more sort heap space than do non-hash join plans. When *sheapthres* is set to be relatively close to *sortheap* (that is, less than a factor of two or three per concurrent query), a hash join runs with much less memory than the optimizer anticipated. When executing with limited memory, hash joins can be very slow. The problem occurs in queries having multiple sorts and hash joins, in which the sorts or hash joins acquire most of the available memory.

The solution is to configure *sheapthres* to be large enough (relative to *sortheap*).

Outer versus Inner Determination

When joining, how are the inner and outer tables determined? The following are general guidelines for how the optimizer decides which table will be the inner and which will be the outer.

In the case of a **hash join**, the inner table is kept in memory buffers. If there are too few memory buffers, then the hash join is obliged to spill. The optimizer attempts to avoid this and so will pick the smaller of the two tables as the inner table, and the larger one as the outer table.

The order in which the tables are accessed is particularly important for a **nested loop join** because the outer table is accessed once but the inner table is accessed once for each row of the outer table. The optimizer chooses the outer and inner tables based on cost estimates. These cost estimates are influenced by the following factors:

- Size

  The smaller table is often chosen to be the outer table to reduce the number of times the inner table must be re-accessed. However, prefetch can cause just the opposite to be true. Prefetching can reduce the cost of accessing a large table substantially. However, usually prefetching is only effective for the outer table of a join. Therefore, the larger table may be accessed first. See “Prefetching Data into the Buffer Pool” on page 241 for more information.
- Predicates

  A table is more likely to be chosen as the outer table if selective predicates can be applied to it because the inner table is only accessed for rows which satisfy the predicates applied to the outer table.
- Buffering

  If the entire inner table must be scanned for each row of the outer table (that is, an index lookup cannot be performed on the inner table), the smaller of the two tables may be chosen as the inner table to take advantage of buffering. This will be influenced by table size and buffer pool size. Note that since join decisions are influenced by buffer pool size, the access plan for your applications may change, if you rebind your applications to the database, after changing the buffer pool size.

  Your ability to create more than one buffer pool, and change the size of that buffer pool, and control the table spaces that use that buffer pool, can affect when buffering is used within inner and outer tables.
- Indexes

  If it is possible to do an index lookup on one of the tables, then that table is a good candidate to use as the inner table. It could then be accessed with an index key lookup using the outer table’s join key predicate as one of the key values. If a table does not have an index, it would not be a good candidate for the inner table since in that case the entire inner table would have to be scanned for every row of the outer table.
- Order requirements

  The table associated with a required order might be assessed first. For example, if the output of the join between t1 and t2 was to be ordered on t1.c, accessing t1 as the outer with an index on t1.c might be a good choice. The output of the join would be ordered and no sort would be required.

  ```
  SELECT * FROM t1, t2
    WHERE t1.a = t2.b
    ORDER BY t1.c
  ```

The order in which the tables are accessed is somewhat less important for a **merge join** because both the inner and outer tables are read only once. However, portions of the inner table which correspond to duplicate join values in the outer are kept in an in-memory buffer. The buffer is reread if the next outer row is the same as the previous outer row, otherwise the buffer is reset. If the number of duplicate join values exceeds the capacity of the in-memory buffer, not all of the duplicates are kept. This will only happen when the duplication on any value is large and the value has a matching value in the outer table.

With all of these considerations for duplicate values, in most cases it is the table with fewer duplicates that will be chosen as the outer table in a join. Ultimately, however, the optimizer chooses the outer and inner tables based on detailed cost estimates.

Search Strategies for Selecting Optimal Join

The optimizer can determine optimal join methods using different search strategies. The search strategy that will be used is determined by the optimization class in use (see "Adjusting the Optimization Class" on page 64). The search strategies and their characteristics are:

- Greedy join enumeration
  - Efficient with respect to space and time
  - Single direction enumeration; that is, once a join method is selected for two tables, it will not be changed during further optimization
  - May miss best access plan when joining many tables. If your query only joins two or three tables, the access plan chosen by the greedy join enumeration will be the same as the access plan chosen by dynamic programming join enumeration. This is particularly true if the query has many join predicates (either explicitly specified, or implicitly generated through predicate transitive closure) on the same column.
- Dynamic programming join enumeration
  - Space and time requirements grow exponentially larger as the number of tables being joined increases
  - Efficient and exhaustive search for best access plan
  - Similar to strategy used by DB2 for OS/390.

The join enumeration algorithm is a key determinant of the number of plan combinations that are explored by the optimizer.

Search Strategies for Star Join

In general, the tables referenced in a query should be connected by join predicates. If two tables are joined without the presence of a join predicate, the Cartesian product of the two tables is formed. That is, every qualifying row of the first table is joined with every qualifying row of the second, creating a result table consisting of the cross product of the size of the two tables that is typically very large. Since such a plan is unlikely to perform very well, the optimizer avoids even determining the cost of such an access plan. The only exception to this occurs when the optimization class is set to 9, or the following special case for “Star Schemas”. For more information, see “Adjusting the Optimization Class” on page 64.

The cases where access plans involving Cartesian products perform well are usually large decision support databases designed with the Star Schema technique. The star schema is a database design in which the bulk of the raw data is kept in a single large table with many columns and is commonly known as a “fact” table. Many of the columns contain encoded values that characterize the dimensions of the particular datum stored in the fact table. In order to allow easy analysis of some subset of the facts, dimension tables are used to decode the encoded values. A typical query would consist of multiple local predicates referencing decoded values in the dimension tables and would contain join predicates connecting the dimension tables to the fact table. For these kinds of queries it may be beneficial to perform the Cartesian product of multiple small dimension tables before accessing the large fact table. This technique is beneficial when multiple join predicates match a multi-column index.

DB2 has the ability to recognize queries against databases designed with star schemas having at least two dimension tables, and to increase the search space to include potential plans that involve forming the Cartesian product of dimension tables. If the plan involving the Cartesian products has the lowest estimated cost, it will be selected by the optimizer.

The Star Schema technique discussed above assumed that primary key indexes were used in the join. Another scenario could involve foreign key indexes. Given that the foreign key columns in the fact table are single-column indexes and that there is a relatively high selectivity across all dimension tables, the following Star Join technique can be used:

1. Each dimension table is processed by:
   - Performing a semi-join between the dimension table and the foreign key index on the fact table
   - Hashing the row ID (RID) values to dynamically create a bitmap.
2. Each bitmap is used with “and” predicates against the previous bitmap (see “Multiple Index Access” on page 159).
3. Determine the surviving RIDs after processing the last bitmap.
4. Optionally sort these RIDs.
5. Fetch a base table row.
6. Re-join the fact table with each of its dimension tables, accessing the dimension tables’ columns that are needed for the SELECT clause

7. Reapply the predicates (residual predicates)

Using this technique, there is no requirement to have multi-column indexes. Explicit referential integrity constraints between the fact table and the dimension tables are not required for this technique to be chosen, although the relationship between the fact table and the dimension tables should have this characteristic.

Composite Tables

Another important parameter determines the shape of the sequence of joins in a query. The result of joining a pair of tables is a new table known as a composite. Typically, this resulting composite table becomes the outer table of a join with another inner table. This is known as a “composite outer”. In some situations, particularly when using the greedy join enumeration technique, it is useful to take the result of joining two tables and make that the inner table of a later join. When the inner table of a join itself consists of the result of joining two or more tables, we say that the plan contains a “composite inner”. For example, in the following query:

```
SELECT COUNT(*)
FROM T1, T2, T3, T4
WHERE T1.A = T2.A AND
      T3.A = T4.A AND
      T2.Z = T3.Z
```

it may be beneficial to join table T1 and T2 (T1xT2), then join T3 to T4 (T3xT4) and finally select the first join result as the outer and the second join result as the inner. In the final plan ((T1xT2) x (T3xT4)) the join result (T3xT4) is known as a composite inner. Depending on the query optimization class, the optimizer places different constraints on the maximum number of tables that may be the inner table of a join. Composite inners are allowed with optimization classes 5, 7, and 9.

Replicated Summary Tables

By using replicated summary tables in a partitioned database environment, you can improve performance by having the database manage pre-computed values of the base table data. For example, the query below would benefit from creating the replicated summary table below. The following assumptions are made:

- The SALES table is in the multipartition table space REGIONTABLESPACE, and is partitioned on the REGION column.
- The EMPLOYEE and DEPARTMENT tables are in a single-partition nodegroup.

You then create a replicated summary table based on the information in the EMPLOYEE table.

```
  CREATE TABLE R_EMPLOYEE
    AS (
        SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
```

```
      FROM   EMPLOYEE
     )
  DATA INITIALLY DEFERRED REFRESH IMMEDIATE
  IN REGIONTABLESPACE
  REPLICATED;
```

Once created, the replicated summary table has its content updated by running this statement:

```
  REFRESH TABLE R_EMPLOYEE;
```

The following example calculates sales by employee, the total for the department, and the grand total:

```
  SELECT d.mgrno, e.empno, SUM(s.sales)
  FROM   department AS d, employee AS e, sales AS s
  WHERE  s.sales_person = e.lastname
     AND e.workdept = d.deptno
  GROUP BY ROLLUP(d.mgrno, e.empno)
  ORDER BY d.mgrno, e.empno;
```

Instead of using the EMPLOYEE table, which is on only one database partition, the database manager will use the R_EMPLOYEE table, which is replicated on each of the database partitions that the SALES tables is on. The performance enhancement occurs because the employee information does not have to be moved across the network to each database partition to calculate the join.

Join Strategies in a Partitioned Database

The following sections describe the join strategies that are possible in a partitioned database environment. The DB2 optimizer automatically selects the best join strategy depending on the requirements of each application. The join strategies are presented here to help you understand what is happening in each strategy. A “table queue” is a mechanism for transferring rows between database partitions, or between processors in a single partition database.

In the descriptions that follow, a *directed* table queue is one whose rows are hashed to one of the receiving database partitions. A *broadcast* table queue is one whose rows are sent to all of the receiving database partitions (that is, it is not hashed). In the diagrams for this section q1, q2, and q3 refer to table queues in the examples. Also the tables that are referenced are divided across two database partitions for the purpose of these scenarios. The arrows indicate the direction in which the table queues are sent. The coordinator node is partition 0.

One consideration for those tables involved in frequent joins in a partitioned database is that of table collocation. Table collocation provides the means in a partitioned database to locate data from one table with the data from another table at the same partition based on the same partitioning key. Once collocated, data to be joined can participate in a query without having to be moved to another database partition as part of the query activity. Only the answer set for the join is moved to the coordinator node. Refer to “Table Collocation” in the *Administration Guide: Planning* for more information on this subject.

For information on join dependencies, refer to the *SQL Reference* manual.

Collocated Joins

For the optimizer to consider a collocated join, the joined tables must be collocated, and all pairs of the corresponding partitioning key must participate in the equijoin predicates. An example is shown in Figure 14.

Note: Replicated summary tables enhance the likelihood of collocated joins. See “Replicated Summary Tables” on page 172for more information.

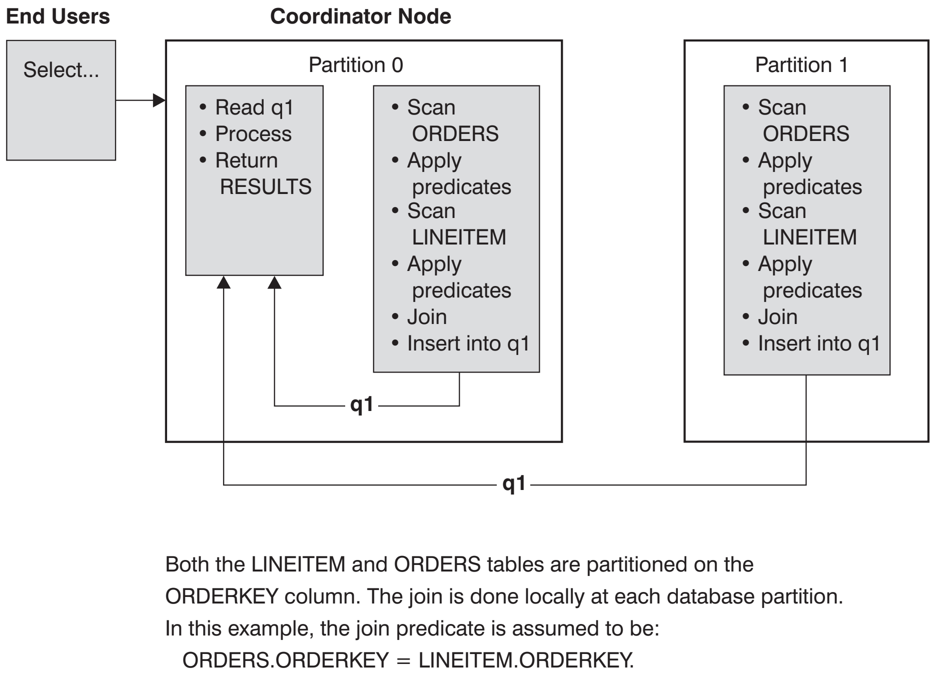
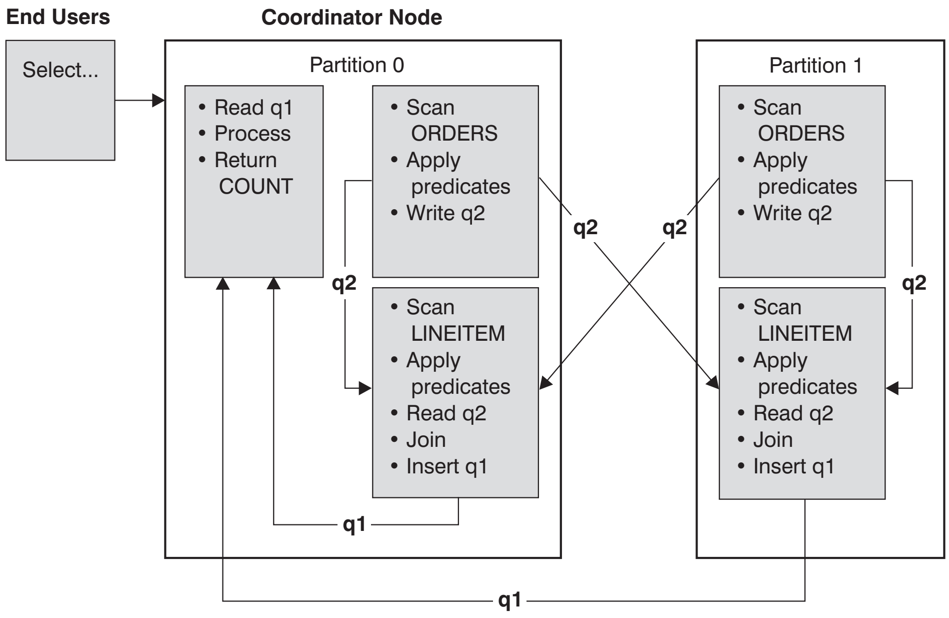


Figure 14. Collocated Join Example

Broadcast Outer-Table Joins

This parallel join strategy can be used if there are no equijoin predicates between the joined tables. It can also be used in other situations in which it is the most cost-effective join method. Typically, this would occur when there is one very large table and one very small table, neither of which is partitioned on the join predicate columns. Rather than partition both tables, it may be "cheaper" to broadcast the smaller table to the larger table. An example is shown in Figure 15.

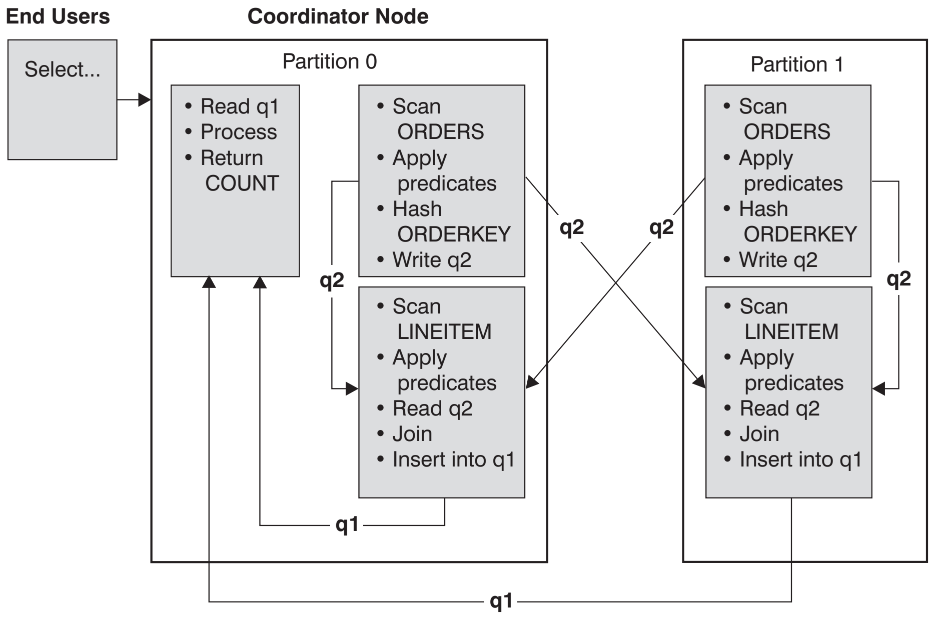


The ORDERS table is sent to all database partitions that have the LINEITEM table. Table queue q2 is broadcast to all database partitions of the inner table.

Figure 15. Broadcast Outer-Table Join Example

Directed Outer-Table Joins

In this join strategy, each row of the outer table is sent to one database partition of the inner table (based on the partitioning attributes of the inner table). The join occurs on this database partition. An example is shown in Figure 16 on page 176.

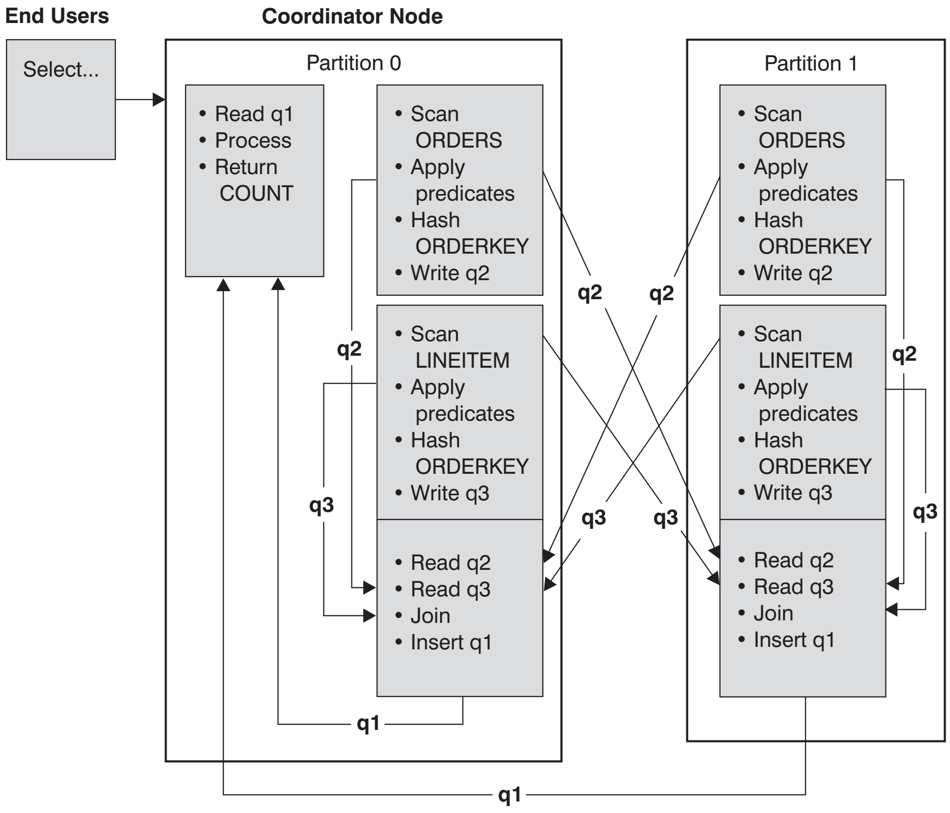


The LINEITEM table is partitioned on the ORDERKEY column.
The ORDERS table is partitioned on a different column.
The ORDERS table is hashed and sent to the correct LINEITEM table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 16. Directed Outer-Table Join Example

Directed Inner-Table and Outer-Table Joins

With this strategy, rows of the outer and inner tables are directed to a set of database partitions, based on the values of the joining columns. The join occurs on these database partitions. An example is shown in Figure 17 on page 177.

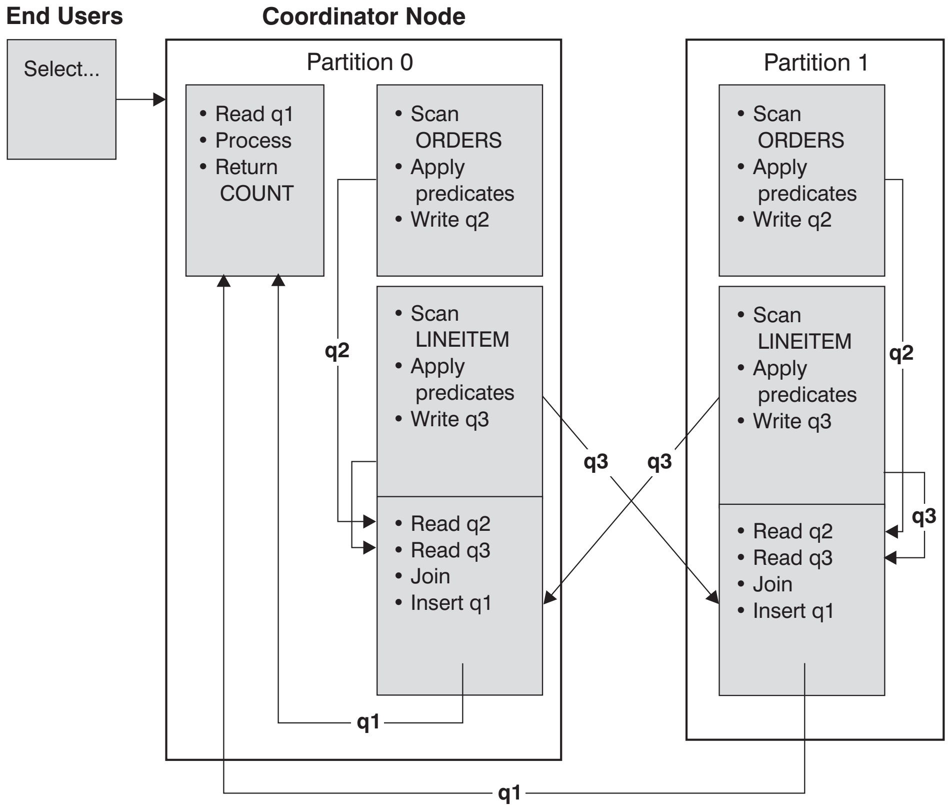


Neither table is partitioned on the ORDERKEY column.
Both tables are hashed and are sent to new database partitions where they are joined.
Both table queue q2 and q3 are directed.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY

Figure 17. Directed Inner-Table and Outer-Table Join Example

Broadcast Inner-Table Joins

With this strategy, the inner table is broadcast to all the database partitions of the outer join table. An example is shown in Figure 18 on page 178.

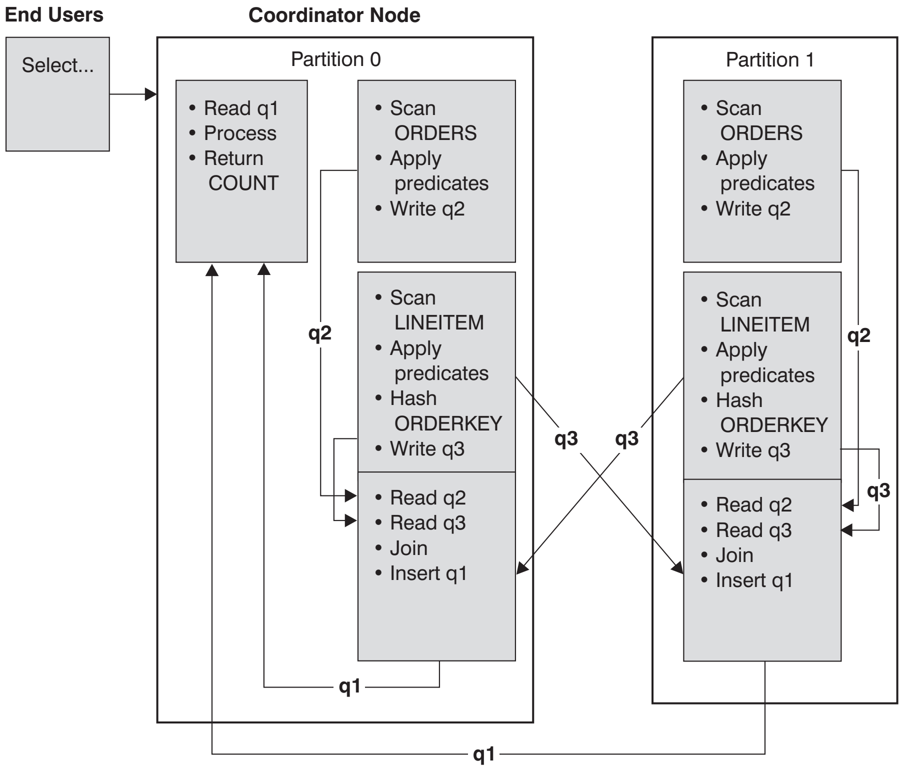


The LINEITEM table is sent to all database partitions that have the ORDERS table.
Table queue q3 is broadcast to all database partitions of the outer table.

Figure 18. Broadcast Inner-Table Join Example

Directed Inner-Table Joins

With this strategy, each row of the inner table is sent to one database partition of the outer join table (based on the partitioning attributes of the outer table). The join occurs on this database partition. An example is shown in Figure 19 on page 179.



The ORDERS table is partitioned on the ORDERKEY column.
The LINEITEM table is partitioned on a different column.
The LINEITEM table is hashed and sent to the correct ORDERS table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 19. Directed Inner-Table Join Example

Table Queues

A table queue is used:

- To pass table data from one database partition to another when using inter-partition parallelism
- To pass table data within a database partition when using intra-partition parallelism
- To pass table data within a database partition when using a single partition database.

Each table queue is used to pass the data in a single direction.

The compiler decides where table queues are required, and includes them in the plan. When the plan is executed, the connections between the database partitions initiate the table queues. The table queues close as processes end.

There are several types of table queues:

- *Asynchronous table queues.* These table queues are known as asynchronous because they read rows in advance of any FETCH being issued by the application. When the FETCH is issued, the row is retrieved from the table queue.

  Asynchronous table queues are used when you specify the FOR FETCH ONLY clause on the SELECT statement. If you are only fetching rows, the asynchronous table queue is faster.
- *Synchronous table queues.* These table queues are known as synchronous because they read one row for each FETCH that is issued by the application. At each database partition, the cursor is positioned on the next row to be read from that database partition.

  Synchronous table queues are used when you do not specify the FOR FETCH ONLY clause on the SELECT statement. In a partitioned database environment, if you are updating rows, the database manager will use the synchronous table queues.
- *Merging table queues.* These table queues preserve order.
- *Non-merging table queues.* These table queues are also known as “regular” table queues. They do not preserve order.
- *Listener table queues.* These table queues are use with correlated subqueries. Correlation values are passed down to the subquery and the results are passed back up to the parent query block using this type of table queue.

Influence of Sorting on the Optimizer

When the optimizer chooses an access plan, it considers the performance impact of sorting data. Sorting occurs when no index exists to satisfy the requested ordering of fetched rows. Sorting could also occur when the sort is determined by the optimizer to be less expensive than an index scan. The optimizer may carry out one of the following actions when sorting the data:

- “Piping” the results of the sort when the query is executed. See “Piped versus Non-Piped Sorts” and “Configuration Parameters Affecting Query Optimization” on page 87.
- Internal handling of the sort within the database manager. See “Aggregation and Sort Pushdown Operators” on page 181.

Piped versus Non-Piped Sorts

At the completion of a sort, if the final sorted list of data can be read in a single sequential pass, the results can be *piped*. Piping is quicker than the use of other (non-piped) means of communicating the results of the sort. The optimizer chooses to pipe the results of a sort whenever possible.

Independent of whether a sort is piped, the time to sort will depend on a number of factors, including the number of rows to be sorted, the key size and the row width. If the rows to be sorted occupy more than the space available in the sort heap, several sort passes are performed, where each pass sorts a subset of the entire set of rows. Each sort pass is stored in a temporary table in the buffer pool. (As part of the buffer pool management, it is possible that pages from this temporary table may be written to disk.) Once all the sort passes are complete, these sorted subsets must be merged into a single sorted set of rows. If the sort is piped, as the rows are merged they are handed directly to Relational Data Services.

For more information, see "Looking for Indicators of Sorting Performance Problems" on page 249, or the discussion of the *sortheap* configuration parameter in "Configuration Parameters Affecting Query Optimization" on page 87.

Aggregation and Sort Pushdown Operators

In some cases, the optimizer can choose to pushdown a sort or aggregation operation to the Data Management Services component from the Relational Data Services component. Pushing down these operations improves performance by allowing the Data Management Services component to pass data directly to a sort or aggregation routine. Without this pushdown, Data Management Services would first pass this data to Relational Data Services, which would then interface with the sort or aggregation routines. For example, the following query benefits from this optimization:

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_DEPT_SALARY
  FROM EMPLOYEE
  GROUP BY WORKDEPT
```

Aggregation in Sort

When sorting is used to produce the order required for a GROUP BY operation the optimizer has the option of performing some or all of the GROUP BY's aggregation while doing the sort. This is advantageous if the number of rows in each group is large. It is even more advantageous if doing some of the grouping during the sort reduces or eliminates the need for the sort to spill to disk.

When aggregation in sort is used, there are up to three (3) stages of aggregation required to ensure proper results are calculated. The first stage of aggregation, "partial aggregation," calculates the aggregate values until the sort heap is filled. Partial aggregation is the process whereby unaggregated data is taken in and partial aggregates are produced. If the sort heap is filled, the rest of the data is spilled to disk and includes all of the partial aggregations that have been calculated in the current filling of the sort heap. Following the reset of the sort heap, new aggregations are started.

The second stage of aggregation, “intermediate aggregation,” takes all of the spilled sort runs, and aggregates further on the grouping keys. The aggregation cannot be completed because the grouping key columns are a subset of the partitioning key columns. Intermediate aggregation takes in existing partial aggregates and produce new partial aggregates. This stage is optional, and is used for both intra-partition parallelism, and for inter-partition parallelism. In the last case, the grouping is finished when a global grouping key is available. In inter-partition parallelism, this would occur when the grouping key is a subset of the partitioning key dividing groups across partitions, and thus requiring repartitioning to complete the aggregation. A similar case exists in intra-partition parallelism when each agent finishes merging it’s spilled sort runs before reducing to a single agent to complete the aggregation.

The last stage of aggregation, “final aggregation,” takes all of the partial aggregates and completes the aggregation. Final aggregation takes in partial aggregates and produces final aggregates. This step always takes place in a GROUP BY operator. Sort cannot do complete aggregation because there is no way to guarantee that the sort will not split. Complete aggregation takes in unaggregated data and produces final aggregates. This method of aggregation is typically used when grouping data that is already in the correct order and when partitioning does not prohibit it’s use.

Optimization Strategies for Intra-Partition Parallelism

The optimizer may choose an access plan so that a query is executed in parallel within a database partition if a degree of parallelism is specified when the SQL statement is compiled.

At execution time, multiple database agents called “subagents” are created to execute the query. The number of subagents is less than or equal to the degree of parallelism determined when the SQL statement was compiled. For more information on setting the degree of parallelism for SQL statements see “Parallel Processing of Applications” on page 84. For more information on agents and subagents, see “Database Agents” on page 257.

In a partitioned database, the degree of parallelism applies to each partition. For example, the portion of the query that is executing at a given database partition is further parallelized based on the degree of parallelism determined at that database partition for that SQL statement.

The access plan is parallelized by dividing it into a portion that is run by each subagent and a portion that is run by the coordinating agent. The subagents pass data through table queues to the coordinating agent or to other subagents. In a partitioned database, subagents may send or receive data through table queues from subagents in other database partitions.

This section describes parallelization strategies within a single database partition.

Parallel Scan Strategies

Relational scans and index scans can be performed in parallel on the same table or index. For parallel relational scans, the table is divided into ranges of pages or rows. A range of pages or rows is assigned to a subagent. A subagent scans its assigned range and is assigned another range when it has completed its work on the current range.

For parallel index scans, the index is divided into ranges of records based on index key values and the number of index entries for a key value. The parallel index scan proceeds like the parallel table scan with subagents being assigned a range of records. A subagent is assigned a new range when it has complete its work on the current range.

The scan unit (either a page or a row) and the scan granularity are determined by the optimizer.

The parallel scan provides an even distribution of work among the subagents. The goal of the parallel scan is to balance the load among the subagents and keep them equally busy. If the number of busy subagents equals the number of available processors and the disks are not overworked with I/O requests, then the machine resources are being used effectively.

Other access plan operations may cause data imbalance as the query executes. The optimizer chooses parallel strategies so that data balance is maintained.

Parallel Sort Strategies

The optimizer may choose one of the following parallel sort strategies:

Round-robin Sort

This is also known as a “redistribution sort”. This is an efficient shared memory sort that attempts to redistribute the data as evenly as possible to all subagents. It uses a round-robin clock type algorithm to provide the even distribution. It first creates an individual sort for each subagent. During the insert phase, subagents insert into each of the individual sorts in a round-robin fashion. This achieves a more even distribution of data.

Partitioned Sort

This is similar to the round-robin sort in that a sort is created for each subagent. The subagents apply a hash function to the sort columns to determine into which sort a row should be inserted. For example, if the inner and outer of a merge join are a partitioned sort, a subagent can use merge join to join the corresponding partitions. This allows the merge join to execute in parallel.

Replicated Sort

This sort is used where all subagents require all the sort output. One sort is created and subagents are synchronized during insertion into the sort. When the sort is completed, each subagent reads the entire sort. This sort may be used to rebalance the data stream if the number of rows is small.

Shared Sort

This sort is the same as a replicated sort, except the subagents open a parallel scan on the sorted result. This distributes the data among the subagents in a way similar to the round-robin sort.

Parallel Temporary Tables

Subagents can cooperate to produce a temporary table by inserting rows into the same table. This is called a `shared temporary table`. The subagents can open private scans or parallel scans on the shared temporary table depending on whether the data stream is to be replicated or partitioned.

Parallel Aggregation Strategies

Aggregation operations can be performed in parallel by subagents. An aggregation operation requires the data to be ordered on the grouping columns. If a subagent can be guaranteed to receive all the rows for a set of grouping column values, it can perform a complete aggregation. This can happen if the stream is already partitioned on the grouping columns because of a previous partitioned sort.

Otherwise the subagent can perform a partial aggregation and use another strategy to complete the aggregation. Some of these strategies are:

- Send the partially aggregated data to the coordinator agent through a merging table queue. The coordinator completes the aggregation.
- Insert the partially aggregated data into a partitioned sort. The sort is partitioned on the grouping columns. This guarantees that all rows for a set of grouping columns are contained in one sort partition.
- If the stream needs to be replicated for balance reasons, the partially aggregated data can be inserted into a replicated sort. Each subagent completes the aggregation using the replicated sort, and receives an identical copy of the aggregation result.

Parallel Join Strategies

Join operations can be performed in parallel by subagents. Parallel join strategies are determined by the characteristics of the data stream.

A join can be parallelized by partitioning and/or replicating the data stream on the inner and outer of the join. For example, a nested loop join can be parallelized if its outer stream is partitioned due to a parallel scan and the inner stream is reevaluated independently by each subagent. A merged join can be parallelized if its inner and outer streams are value-partitioned due to partitioned sorts.

Automatic Summary Tables

Summary tables are a powerful way to improve query response time. In many environments where some of the basic query structures can be anticipated summary tables can be used to:

- Aggregate data over one or more dimensions
- Join and aggregate data over a group of tables
- Identify a commonly accessed subset of data (that is, a “hot” horizontal or vertical partition)
- Repartition a table, or part of a table, in a partitioned database environment

Knowledge of summary tables is integrated into the SQL Compiler. Within the SQL Compiler, Query Rewrite (see “Rewrite Query by the SQL Compiler” on page 143) and the Optimizer (see “Data Access Concepts and Optimization” on page 153) are involved in matching queries with summary tables in and determining whether to substitute a summary table for a query over base tables. Whenever summary tables are used to answer queries the EXPLAIN facilities (see “Chapter 7. SQL Explain Facility” on page 201) can be used to determine which summary table was selected. Since summary tables behave like regular tables in many ways, the same considerations for optimizing data access using tablespace definitions, creating indexes, and issuing RUNSTATS apply to summary tables.

To help you understand the power of summary tables we provide the following example of a multidimensional analysis query and show how it takes advantage of summary tables.

In this example, we assume a scenario where a warehouse contains a set of customers and a set of credit card accounts. The warehouse records the set of transactions that are made with the credit cards. Each transaction contains a set of items that are purchased together. We can categorize this environment as a multi-star because two tables, the one containing transaction items and the other identifying the purchase transactions, are large and together are the hub of the star.

There are three hierarchical dimensions that describe a transaction: product, location, and time. The product hierarchy is recorded in two normalized tables representing the product group and the product line. The location hierarchy contains city, state, and country information and is represented in a single de-normalized table. The time hierarchy contains day, month, and year information and is encoded in a single date field. The date dimensions are extracted from the date field of the transaction using built-in functions. There are also other tables in this scenario that represent account information for customers and customer information.

A summary table is created with the sum and count of sales for each level of:
- Product hierarchy
- Location hierarchy
- Time hierarchy, composed of year, month, day.

A wide range of queries can pick up their answers from this stored aggregate data. The following example computes sum and count of sales along the product group and line dimensions; along the city, state, and country dimension; and along the time dimension. It also includes several other columns in its GROUP BY clause.

```
CREATE TABLE dba.PG_SALESSUM
  AS (
      SELECT l.id AS prodline, pg.id AS pgroup,
             loc.country, loc.state, loc.city,
             l.name AS linename, pg.name AS pgname,
             YEAR(pdate) AS year, MONTH(pdate) AS month,
             t.status,
             SUM(ti.amount) AS amount,
             COUNT(*) AS count
      FROM   cube.transitem AS ti, cube.trans AS t,
             cube.loc AS loc, cube.pgroup AS pg,
             cube.prodline AS l
      WHERE  ti.transid = t.id
         AND ti.pgid = pg.id
         AND pg.lineid = l.id
         AND t.locid = loc.id
         AND YEAR(pdate) > 1990
      GROUP BY l.id, pg.id, loc.country, loc.state, loc.city,
               year(pdate), month(pdate), t.status, l.name, pg.name
     )
DATA INITIALLY DEFERRED REFRESH DEFERRED;

REFRESH TABLE dba.SALESCUBE;
```

The summary table is typically much smaller than the base fact tables. You can control when the summary table is refreshed by specifying the DEFERRED option (as shown in our example).

Queries that can take advantage of such pre-computed sums would include:
- Sales by month and product group
- Total sales for years after 1990
- Sales for 1995 or 1996
- Sum of sales for a product group or product line
- Sum of sales for a specific product group or product line AND for 1995, 1996

- Sum of sales for a specific country.

While the precise answer is not included in the summary table for any of these queries, the cost of computing the answer using the summary table could be significantly less than using a large base table, because a portion of the answer is already computed. Expensive joins, sorts, and aggregation of base data is avoided or reduced through summary tables.

The following are sample queries that would obtain significant performance improvements because they are able to use the results in the summary table that are already computed. The first example returns the total sales for 1995 and 1996:

```
SET CURRENT REFRESH AGE=ANY

SELECT YEAR(pdate) AS year, SUM(ti.amount) AS amount
FROM   cube.transitem AS ti, cube.trans AS t,
       cube.loc AS loc, cube.pgroup AS pg,
       cube.prodline AS l
WHERE  ti.transid = t.id
   AND ti.pgid = pg.id
   AND pg.lineid = l.id
   AND t.locid = loc.id
   AND YEAR(pdate) IN (1995, 1996)
GROUP BY year(pdate);
```

The second example returns the total sales by product group for 1995 and 1996:

```
SET CURRENT REFRESH AGE=ANY

SELECT pg.id AS "PRODUCT GROUP",
       SUM(ti.amount) AS amount
FROM   cube.transitem AS ti, cube.trans AS t,
       cube.loc AS loc, cube.pgroup AS pg,
       cube.prodline AS l
WHERE  ti.transid = t.id
   AND ti.pgid = pg.id
   AND pg.lineid = l.id
   AND t.locid = loc.id
   AND YEAR(pdate) IN (1995, 1996)
GROUP BY pg.id;
```

Larger improvements in response time for such queries can be achieved with larger databases. This happens because the summary table grows slower than the growth of the base table. One advantage of summary tables is that DB2 Universal Database uses them to effectively eliminate overlapping work among queries by doing the computation once when building the summary tables and reusing their content for a very large number of queries.

Federated Database Query Compiler Phases

This section describes additional query processing phases in a federated database system. It also provides recommendations for improving federated database query performance. Major topics include:

- "Pushdown Analysis"
- "Remote SQL Generation and Global Optimization" on page 195.

Pushdown Analysis

Pushdown analysis tells the DB2 optimizer if an operation can be performed at a remote data source. An operation can be a function, such as relational operator, system or user functions, or an SQL operator (GROUP BY, ORDER BY, and so on).

Functions that cannot be pushed-down can significantly impact query performance. Consider the effect of forcing a selective predicate to be evaluated locally instead of at the data source. This approach could require DB2 to retrieve the entire table from the remote data source and then filter it locally against the predicate. If your network is constrained—and the table is large—query performance could suffer.

Operators that are not pushed-down can also significantly impact query performance. For example, having a GROUP BY operator aggregate remote data locally could, once again, require DB2 to retrieve the entire table from the remote data source.

As an example, assume that nickname N1 references the data source table EMPLOYEE in a DB2 for OS/390 data source. Further, assume that the table has 10,000 rows, one of the columns contains the last names of employees, and one of the columns contains salaries. Given the statement:

```
SELECT LASTNAME, COUNT(*)  FROM N1
   WHERE LASTNAME  > 'B' AND SALARY > 50000
   GROUP BY LASTNAME;
```

several possibilities are considered:

- If the collating sequences at DB2 and DB2 for OS/390 are the same, it is likely that the query predicate will be pushed-down to DB2 for OS/390. It is usually more efficient to filter and group results at the data source instead of copying the entire table to DB2 and performing the operations locally. Pushdown analysis in federated systems determines if operations can be performed at the data source. In this case, the predicate and the GROUP BY operation can take place at the data source.
- If the collating sequence is not the same, pushdown analysis will determine that the entire predicate cannot be evaluated at the data source; however, the optimizer may decide to pushdown the `SALARY > 1000` portion of the predicate. The range comparison must still be done at DB2.

- If the collating sequence is the same, and the optimizer knows that the local DB2 server is very fast, it is possible that the optimizer will decide that performing the `GROUP BY` operation locally at DB2 is the best (least cost) approach. The predicate will be evaluated at the data source. This is an example of pushdown analysis combined with global optimization. DB2 will consider the available paths and then choose a plan that is the most efficient.

In general, the goal is to ensure that functions and operators can be considered for evaluation on data sources by the optimizer. Many factors can affect whether a function or an SQL operator is evaluated at a remote data source. The key factors are discussed in three groups: server characteristics, nickname characteristics, and query characteristics.

Server Characteristics Affecting Pushdown Opportunities

The following sections contain data source-specific factors that can affect pushdown opportunities. In general, these factors exist because DB2 lets you use a rich SQL dialect to submit queries. This dialect may offer more functionality than the SQL dialect supported by a server accessed during a DB2 query. DB2 can compensate for the lack of function at a data server, but doing so may require that the operation take place at DB2.

SQL Capabilities: Each data source supports a variation of the SQL dialect and different levels of functionality. For example, consider the GROUP BY list. Most data sources support the GROUP BY operator; but, some have restrictions on the number of items on the GROUP BY list or restrictions on whether an expression is allowed on the GROUP BY list. If there is a restriction at the remote data source, DB2 might have to perform the GROUP BY operation locally.

SQL Restrictions: Each data source can have different SQL restrictions. For example, some data sources require parameter markers to bind in values to remote SQL statements. Therefore, parameter marker restrictions must be checked to ensure that each data source can support such a bind mechanism. If DB2 cannot determine a good method to bind in a value for a function, this function must be evaluated locally.

SQL Limits: DB2 might allow the use of larger integers than its remote data sources; however, limit-exceeding values cannot be embedded in statements sent to data sources. Therefore, the function or operator that operates on this constant must be evaluated locally.

Server Specifics: Several factors fall into this category. One example is sorting NULL values (highest, lowest, or depending on the ordering). For example, if the NULL value is sorted at a data source differently from DB2, ORDER BY operations on a nullable expression cannot be remotely evaluated.

Collating Sequence: Configuring a federated database to use the same collating sequence that a data source uses and then setting the collating_sequence server option to ′Y′ allows the optimizer to consider ″pushing-down″ character range comparison predicates.

When a query from a federated server requires sorting, the place where the sorting is processed depends on several factors. If the federated database's collating sequence is the same as that of the data source where the queried data is stored, the sort may take place at the data source. If collating sequences are the same, the optimizer can decide if a local sort or a sort at the data source is the most efficient way to complete the query. Likewise, if a query requires a comparison of character data, this comparison can also be performed at the data source.

Numeric comparisons, in general can be done at either location even if the collating sequence is different. You may get unusual results, however, if the weighting of null characters is different between the federated database and the data source. Likewise, for comparison statements, be careful if you are submitting statements to a case-insensitive data source. The weights assigned to the characters ″I″ and ″i″ in a case-insensitive data source are the same. DB2, by default, is case sensitive and would assign different weights to the characters.

If the collating sequences of the federated database and the data source differ, DB2 retrieves the data to the federated database, so that it can do the sorting and comparison locally. The reason is that users expect to see the query results ordered according to the collating sequence defined for the federated server; by ordering the data locally, the federated server ensures that this expectation is fulfilled.

Retrieving data for local sorts and comparisons usually decreases performance. Therefore, consider configuring the federated database to use the same collating sequences that your data sources use. That way, performance might increase, because the federated server can allow sorts and comparisons to take place at data sources. For example, in DB2 UDB for OS/390, sorts defined by ORDER BY clauses are implemented by a collating sequence based on an EBCDIC code page. If you want to use the federated server to retrieve DB2 for OS/390 data sorted in accordance with ORDER BY clauses, it is advisable to configure the federated database so that it uses a predefined collating sequence based on the EBCDIC code page.

If the collating sequences at the federated database and the data source differ, and you need to see the data ordered in the data source's sequence, you can submit your query in pass-through mode, or define the query in a data source view.

See the *Administration Guide: Planning* for more information about collating sequences and how to set them; see Table 8 on page 102 for more information about the collating_sequence server option.

Server Options: Several server options can affect pushdown opportunities. In particular, review your settings for *collating_sequence*, *varchar_no_trailing_blanks*, and *pushdown*. See “Server Options Affecting Federated Database Queries” on page 101 for information on setting these options.

DB2 Type Mapping and Function Mapping Factors: The default local data type mappings provided by DB2 (see the *Application Development Guide* for data type tables) are designed so that sufficient buffer space is given to each data source data type (to avoid loss of data). A user can choose to customize the type mapping for a specific data source to suit specific applications. For example, if you are accessing an Oracle data source column with a DATE data type (which by default is mapped to the DB2 TIMESTAMP data type), you could change the local data type to the DB2 DATE data type.

DB2 can compensate for functions not supported by a data source. There are three cases where function compensation will occur:

- This function simply does not exist at the remote data source.
- The function does exist; however, the characteristics of the operand violate function restrictions. An example of this situation is the IS NULL relational operator. Most data sources support it, but some may have restrictions, such as only allowing a column name on the left hand side of the IS NULL operator.
- The function, if evaluated remotely, may return a different result. An example of this situation is the ’>’ (greater than) operator. For those data sources with different collating sequences, the greater than operator may return different results than if it is evaluated locally by DB2.

Nickname Characteristics Affecting Pushdown Opportunities

The following sections contain nickname-specific factors that can affect pushdown opportunities.

Local Data Type of a Nickname Column: Ensure that the local data type of a column does not prevent a predicate from being evaluated at the data source. As mentioned earlier, the default data type mappings are provided to avoid any possible overflow. However, a joining predicate between two columns of different lengths might not be considered at the data source whose joining column is shorter, depending on how DB2 binds in the longer column. This situation can affect the number of possibilities in a joining sequence evaluated by the DB2 optimizer. For example, Oracle data source columns created using the INTEGER or INT data type are given the type NUMBER(38). A nickname column for this Oracle data type will be given the local data type FLOAT because the range of a DB2 integer is from 2**31 to

(-2**31)-1, which is roughly equal to NUMBER(9). In this case, joins between a DB2 integer column and an Oracle integer column cannot take place at the DB2 data source (shorter joining column); however, if the domain of this Oracle integer column can be accommodated by the DB2 INTEGER data type, change its local data type with the ALTER NICKNAME statement so that the join can take place at the DB2 data source.

Column Options: The ALTER NICKNAME SQL statement can be used to add or change column options for nicknames.

One of these options is ″varchar_no_trailing_blanks″. It can be used to identify a column that contains no trailing blanks. The compiler pushdown analysis step will then take this information into account when checking all operations performed on columns so indicated. Based on this indication, DB2 may generate a different but equivalent form of a predicate to be used in the remote SQL statement sent to a data source. A user might see a different predicate being evaluated against the data source, but the net result should be equivalent.

Another column option is numeric_string. Use this option to indicate if the values in that column are always numbers without trailing blanks.

See Table 15 for column option values and defaults.

Table 15. Column Options and Their Settings

Option	Valid Settings		Default Setting
numeric_string	‘Y’	Yes, this column contains only strings of numeric data. IMPORTANT: If this column contains only numeric strings followed by trailing blanks, it is inadvisable to specify ‘Y’.	‘N’
	‘N’	No, this column is not limited to strings of numeric data.	
	By setting numeric_string to ‘Y’ for a column, you are informing the optimizer that this column contains no blanks that could interfere with sorting of the column’s data. This option is helpful when the collating sequence of a data source is different from DB2. Columns marked with this option will not be excluded from local (data source) evaluation because of a different collating sequence.		

Table 15. Column Options and Their Settings (continued)

Option	Valid Settings	Default Setting
varchar_no_trailing_blanks	Indicates whether trailing blanks are absent from a specific VARCHAR column: ‘Y’ Yes, trailing blanks are absent from this VARCHAR column. ‘N’ No, trailing blanks are not absent from this VARCHAR column. If data source VARCHAR columns contain no padded blanks, then the optimizer’s strategy for accessing them depends in part on whether they contain trailing blanks. By default, the optimizer “assumes” that they actually do contain trailing blanks. On this assumption, it develops an access strategy that involves modifying queries so that the values returned from these columns are the ones that the user expects. If, however, a VARCHAR column has no trailing blanks, and you let the optimizer know this, it can develop a more efficient access strategy. To tell the optimizer that a specific column has no trailing blanks, specify that column in the ALTER NICKNAME statement (for syntax, see the *SQL Reference*).	‘N‘

Query Characteristics Affecting Pushdown Opportunities

A query can reference an SQL operator that might involve nicknames from multiple data sources. When DB2 must combine the results from two referenced data sources using one operator, such as a set operator (e.g. UNION), the operation must take place at DB2. The operator cannot be evaluated at a remote data source directly.

Analyzing and Understanding Pushdown Analysis Decisions

Rewriting SQL statements can provide additional pushdown opportunities for DB2 query processing. This section introduces tools for determining where a query is evaluated, lists common questions (and suggested areas to investigate) associated with query analysis, and closes with a brief section about data source upgrades.

Analyzing Where a Query is Evaluated: There are two utilities provided with DB2 that show where queries are evaluated:

- Visual explain. Start it with the **db2cc** or the **db2vexp** command. Use it to view the query access plan graph. The execution location for each operator is included in the detailed display of an operator.

  If a query is completely pushed down, you should see a RETURN operator on top of an RQUERY operator. The RETURN operator is a standard DB2 operator; the RQUERY operator is unique to federated database operations.

RQUERY sends an SQL SELECT statement to a data source to retrieve the query result. The SELECT statement is generated using the SQL dialect supported by the data source. It can contain any valid query for that data source.

- SQL explain. Start it with the **db2expln** or the **dynexpln** command. Use it to view the access plan strategy as text.

Understanding Why a Query is Evaluated at a Data Source or at DB2: This section lists typical plan analysis questions and areas to investigate to increase pushdown opportunities. Key questions include:

- Why isn't this predicate being evaluated remotely?

  This question arises when a predicate is very selective and thus could be used to filter rows and reduce network traffic. Remote predicate evaluation also affects whether a join between two tables of the same data source can be evaluated remotely.

  Areas to examine include:

  – Subquery predicates. Does this predicate contain a subquery that pertains to another data source? Does this predicate contain a subquery involving an SQL operator that is not supported by this data source? Not all data sources support set operators in a predicate.
  – Predicate functions. Does this predicate contain a function that cannot be evaluated by this remote data source? Relational operators are classified as functions.
  – Predicate bind requirements. Does this predicate, if remotely evaluated, require bind-in of some value? If so, would it violate SQL restrictions at this data source?
  – Global optimization. The optimizer may have decided that local processing is more cost effective. See "Remote SQL Generation and Global Optimization" on page 195 for more information.

- Why isn't the GROUP BY operator evaluated remotely?

  There are several areas you can check:

  – Is the input to the GROUP BY operator evaluated remotely? If the answer is no, examine the input.
  – Does the data source have any restrictions on this operator? Examples include:
    - Limited number of GROUP BY items
    - Limited byte counts of combined GROUP BY items
    - Column specification only on the GROUP BY list
  – Does the data source support this SQL operator?
  – Global optimization. The optimizer may have decided that local processing is more cost effective. See "Remote SQL Generation and Global Optimization" on page 195 for more information.

- Why isn’t the set operator evaluated remotely?

  There are several areas you can check:
  - Are both of its operands completely evaluated at the same remote data source? If the answer is no and it should be yes, examine each operand.
  - Does the data source have any restrictions on this set operator? For example, are large objects or long fields valid input for this specific set operator?
- Why isn’t the ORDER BY operation evaluated remotely?

  Consider:
  - Is the input to the ORDER BY operation evaluated remotely? If the answer is no, examine the input.
  - Does the ORDER BY clause contain a character expression? If yes, does the remote data source not have the same collating sequence as DB2?
  - Does the data source have any restrictions on this operator? For example, is there a limited number of ORDER BY items? Does the data source restrict column specification to the ORDER BY list?

Data Source Upgrades and Customization: Although the DB2 SQL compiler has much information about data source SQL support, this data may need adjustment over time because data sources can be upgraded and/or customized. In such cases, make enhancements known to DB2 by changing local catalog information. Use DB2 DDL statements (such as CREATE FUNCTION MAPPING and ALTER SERVER) to update the catalog. See the *SQL Reference* for more information.

Remote SQL Generation and Global Optimization

This phase helps produce a globally optimal access strategy to evaluate a query. For a federated database query, the access strategy may involve breaking down the original query into a set of remote query units and then combining the results.

Using the output of pushdown analysis as a recommendation, the optimizer decides whether each operation will be evaluated locally at DB2 or remotely at a data source. The decision is based on the output of its cost model, which includes not only the cost to evaluate the operation but also the cost to transmit the data or messages between DB2 and data sources.

The goal is to produce an optimized query; however, many factors can affect the output from global optimization and thus affect query performance. The key factors are discussed in two groups: server characteristics and nickname characteristics.

Server Characteristics/Options Affecting Global Optimization

Data source server factors that can affect global optimization include the:

- Relative ratio of CPU speed

  Use the *cpu_ratio* server option to indicate how much faster or slower the data source CPU speed is compared with the DB2 CPU. A low ratio indicates that the data source workstation CPU is faster than the DB2 workstation CPU. With low ratios, the DB2 optimizer is more likely to consider pushing-down CPU-intensive operations to the data source.See "Server Options Affecting Federated Database Queries" on page 101 for more information about this ratio.
- Relative ratio of I/O speed

  Use the *io_ratio* server option to indicate how much faster or slower the data source system I/O speed is compared with the DB2 system. A low ratio indicates that the data source workstation I/O speed is faster than the DB2 workstation I/O speed. For low ratios, the DB2 optimizer will consider pushing-down I/O-intensive operations to the data source. See "Server Options Affecting Federated Database Queries" on page 101 for more information about this ratio.
- Communication rate between DB2 and the data source

  Use the *comm_rate* server option to indicate network capacity. Low rates (indicating a slow network communication between DB2 and the data source) encourage the DB2 optimizer to reduce the number of messages sent to or from this data source. If the rate is set to 0, the optimizer produces a query requiring minimal network traffic. See "Server Options Affecting Federated Database Queries" on page 101 for more information about this ratio.
- Data source collating sequence

  Use the *collating_sequence* server option to indicate if a data source collating sequence matches the local DB2 database collating sequence. If this option is not set to 'Y', the optimizer considers the data retrieved from this data source as unordered. See "Collating Sequence" on page 190 for more information about collating sequence performance issues.
- Remote plan hints

  Use the *plan_hints* server option to indicate if plan hints are supported at a data source. Plan hints are statement fragments that provide extra information for data source optimizers. This information can, for certain query types, improve query performance. The plan hints can help the data source optimizer decide whether to use an index, which index to use, or which table join sequence to use.

  If plan hints are enabled, the query sent to the data source contains additional information. For example, a statement sent to an Oracle optimizer with plan hints could look like this:

```
SELECT /*+ INDEX (table1, t1index)*/
   col1
   FROM table1
```

The plan hint is the string `/*+ INDEX (table1, t1index)*/`.

- Information in the DB2 optimizer knowledge base

  DB2 has an optimizer knowledge base that contains data about native data sources. The DB2 optimizer does not generate remote access plans that cannot be generated by specific DBMSs. In other words, DB2 avoids generating plans that optimizers at remote data sources cannot understand or accept.

Nickname Characteristics Affecting Global Optimization

The following sections contain nickname-specific factors that can affect global optimization.

Index Considerations: DB2 can use information about indexes at data sources to optimize queries. For this reason, it is important that the index information available to DB2 is current. The index information for nicknames is initially acquired at create nickname time. Index information is not gathered for view nicknames.

Creating Index Specifications on Nicknames: You can create an index specification for a nickname. Index specifications build an index definition (not an actual index) in the catalog for use by the DB2 optimizer. Use the CREATE INDEX SPECIFICATION ONLY statement to create index specifications. The syntax for creating an index specification on a nickname is similar to the syntax for creating an index on a local table. See the *Administration Guide: Planning* for more information.

Consider creating index specifications when:

- DB2 is unable to retrieve any index information from a data source during nickname creation.
- You want an index for a view nickname.
- You want to encourage the DB2 optimizer to use a specific nickname as the inner table of a nested loop join. The user can create an index on the joining column if none exists.

Consider your needs before issuing CREATE INDEX statements against a nickname for a view. In one case, if the view is a simple SELECT on a table with an index, creating indexes on the nickname (locally) that match the indexes on the table at the data source can significantly improve query performance. However, if indexes are created locally over views that are not simple select statements (for example, a view created by joining two tables), query performance may suffer. For example, if an index is created over a view that is a join of two tables, the optimizer may choose that view as the inner element in a nested loop join. The query will have poor performance because the join will be evaluated several times. An alternative is to create nicknames for each of the tables referenced in the data source view and create a local view at DB2 that references both nicknames.

Catalog Statistics Considerations: Catalog statistics describe the overall size of nicknames and the range of values in associated columns. They are used by the optimizer when calculating the least cost path for processing queries containing nicknames. Nickname statistics are stored in the same catalog views as table statistics. See "Chapter 5. System Catalog Statistics" on page 107 and "Rules for Updating Table and Nickname Statistics" on page 130 for more information about statistic types and how to update them locally.

While DB2 can retrieve the statistical data held at a data source, it cannot automatically detect updates to existing statistical data at data sources. Furthermore, DB2 has no mechanism for handling object definition or structural changes (adding a column) to objects at data sources. If the statistical data or structural data for an object has changed, you have two choices:

- Run the equivalent of RUNSTATS at the data source. Then, drop the current nickname. Re-create the nickname. Use this approach if structural information has changed.
- Manually update the statistics in the SYSSTAT.TABLES view. This approach requires fewer steps but it will not work if structural information has changed.

Analyzing and Understanding Global Optimization Decisions

This section introduces tools for analyzing query optimization and presents common questions (and suggested areas to investigate) associated with query optimization.

Analyzing Query Optimization: There are two utilities provided with DB2 that show global access plans:

- Visual explain. Start it with the **db2cc** or the **db2vexp** command. Use it to view the query access plan graph. The execution location for each operator is included in the detailed display of an operator. You can also find the remote SQL statement generated for each data source in the RQUERY (select operation) operator. By examining the details of each operator, you can see the number of rows estimated by the DB2 optimizer as input to and output from each operator. You can also see the estimated cost to execute each operator including the communications cost. See "Appendix C. SQL Explain Tools" on page 531 for more information.
- SQL explain. Start it with the db2expln or dynexpln command. Use it to view the access plan strategy as text. SQL explain does not provide cost information; however, you can get the access plan generated by the remote

optimizer for those data sources supported by the remote explain function. See "Appendix C. SQL Explain Tools" on page 531 for more information.

Understanding DB2 Optimization Decisions: This section lists optimization questions and key areas to investigate to improve performance. Key questions include:

- Why isn't a join between two nicknames of the same data source being evaluated remotely?

  Areas to examine include:
  - Join operations. Can the data source support them?
  - Join predicates. Can the join predicate be evaluated at the remote data source? If the answer is no, examine the join predicate. See "Understanding Why a Query is Evaluated at a Data Source or at DB2" on page 194 for more information.
  - Number of rows in the join result (with visual explain). Does the join produce a much larger set of rows than the two nicknames combined? Do the numbers make sense? If the answer is no, consider updating the nickname statistics manually (SYSSTAT.TABLES).
- Why isn't the GROUP BY operator being evaluated remotely?

  Areas to examine include:
  - Operator syntax. Verify that the operator can be evaluated at the remote data source. See "Understanding Why a Query is Evaluated at a Data Source or at DB2" on page 194 for more information.
  - Number of rows. Check the estimated number of rows in the GROUP BY operator input and output using visual explain. Are these two numbers very close? If the answer is yes, the DB2 optimizer considers it more efficient to evaluate this GROUP BY locally. Also, do these two numbers make sense? If the answer is no, consider updating the nickname statistics manually (SYSSTAT.TABLES).
- Why is the statement not being completely evaluated by the remote data source?

  The DB2 Optimizer performs cost-based optimization. Even if pushdown analysis indicates that every operator can be evaluated at the remote data source, the optimizer still relies on its cost estimate to generate a globally optimal plan. There are a great many factors that can contribute to that plan. For example, even though the remote data source can process every operation in the original query, its CPU speed is much slower than DB2's and thus it may turn out to be more beneficial to perform the operations at DB2 instead. If results are not satisfactory, verify your server statistics in SYSCAT.SERVEROPTIONS.
- Why does a plan generated by the optimizer, and completely evaluated at a remote data source, have much worse performance than the original query executed directly at the remote data source?

Areas to examine include:

– The remote SQL statement generated by the DB2 optimizer. Ensure that it is identical to the original query. Check for predicate ordering changes. A good query optimizer should not be sensitive to the predicate ordering of a query; unfortunately, not all DBMS optimizers are identical, and thus it is likely that the optimizer of the remote data source may generate a different plan based on the input predicate ordering. If this is true, this is a problem inherent in the remote optimizer. Consider either modifying the predicate ordering on the input to DB2 or contacting the service organization of the remote data source for assistance.

  Also, check for predicate replacements. A good query optimizer should not be sensitive to equivalent predicate replacements; unfortunately, not all DBMS optimizers are identical, and thus it is possible that the optimizer of the remote data source may generate a different plan based on the input predicate. For example, some optimizers cannot generate transitive closure statements for predicates.
– The number of returned rows. You can get this number from Visual Explain. If the query returns a large number of rows, network traffic is a potential bottleneck.
– Additional functions. Does the remote SQL statement contain additional functions compared with the original query? Some of the extra functions may be generated to convert data types. Ensure that they are necessary.

Chapter 7. SQL Explain Facility

The SQL explain facility is part of the SQL Compiler that can be used to capture information about the environment where the static or dynamic SQL statement is compiled. The information captured allows you to understand the structure and potential execution performance of SQL statements, including:

- Sequence of operations to process the query
- Cost information
- Predicates and selectivity estimates
- Statistics for all objects referenced in the SQL statement at the time of the explain.

This information can help you:

- Understand the execution plan chosen for a query
- Assist in designing application programs
- Determine when an application should be rebound
- Assist in database design.

The following topics are provided:

The explain output is stored in relational tables and, as an option, in a format which may be graphically displayed using the Visual Explain tool. You should consider using the explain tables to find those queries run against the explain tables that are of interest to you. For more information on the tables used by the explain facility and how to create those tables, see "Appendix B. Explain Tables and Definitions" on page 497.

Choosing an Explain Tool

DB2 provides the most comprehensive explain facility in the industry with detailed optimizer information on the access plan chosen for an explained SQL statement. Several methods are provided to give you the flexibility you need to capture and access explain information.

Detailed optimizer information that allows for in-depth analysis of an access plan is kept in explain tables separate from the actual access plan itself. There are three ways to get information from the explain tables:

1. Write your own queries (based on the explain table descriptions as shown in "Appendix B. Explain Tables and Definitions" on page 497)
2. Use the *db2exfmt* tool
3. Use Visual Explain (to view explain snapshot information)

The explain tables are accessible on all supported platforms and contain information for both static and dynamic SQL statements. You can access the explain tables using SQL statements which allows for easy manipulation of the output and for comparison among different queries, or for comparisons of the same query over time. If you wish the information from the explain tables to be presented in a predefined format, you can use the *db2exfmt* tool. For more information about this tool, see "Appendix D. db2exfmt - Explain Table Format Tool" on page 577. Alternatively, you are required to create your own statements to access the tables.

Note: The location of this tool (and others like *db2batch*, *dynexpln*, *db2vexp*, and *db2_all*) is in the *misc* subdirectory of the *sqllib* directory. If this tool has been moved from this path, then the command line entry mentioned above may not work.

Visual Explain allows for the analysis of access plan and optimizer information from the explain tables through a graphical interface. Both static and dynamic SQL statements can be analyzed using this tool. Visual Explain is typically invoked from within the Control Center. The Control Center is available from the command line by typing *db2cc*. Also, Visual Explain can be invoked directly from the command line for a single SQL statement using the *db2vexp* command. On some platforms, Visual Explain can be invoked using a folder from within the DB2 Universal Database folder. Visual Explain is not available on all supported platforms. You should refer to the *Quick Beginnings* manual for your platform to see if Visual Explain is supported. Visual Explain does allow you to view snapshots captured or taken on another platform. For example, a Windows NT Client can graph snapshots generated on a DB2 for HP-UX server. To do this, both of the platforms must be at a Version 5 level or later. The output from Visual Explain is not easily manipulated for further analysis nor is the information accessible to other applications. For more information on the *db2vexp* command, type *db2vexp -h* on the command line or refer to the *Command Reference* manual. For other information on Visual Explain, you should refer to the online help in the `Control Center` by typing *db2cc*.

Information about access plans for static SQL statements is generated and stored in the system catalog as part of a package. To see the access plan information available for one or more packages, the *db2expln* tool is available from the command line. *db2expln* shows the actual implementation of the chosen access plan. It does not show optimizer information.

The *dynexpln* tool, which uses *db2expln* within it, provides a quick way to explain dynamic SQL statements that contain no parameter markers. This use of *db2expln* from within *dynexpln* is done by transforming the input SQL statement into a static statement within a pseudo-package. When this occurs, the information may not always be completely accurate. If complete accuracy is desired, you should use the Explain facility as described in "Using the SQL Explain Facility" on page 204.

The *db2expln* tool does provide a relatively compact and English-like overview of what operations will occur at run-time by examining the actual access plan generated (see 142 for more information on how the code is generated). Additional details on using *db2expln* and interpreting the output can be found in "Appendix C. SQL Explain Tools" on page 531.

Table 16 summarizes the different tools available with the DB2 explain facility and their individual characteristics. Use this table to select the tool most suitable for your environment and needs.

Table 16. Explain Facility Tools

Desired Characteristics	**Visual Explain**	**db2vexp**	**Explain tables**	**db2exfmt**	**db2expln**	**dynexpln**
GUI-interface	Yes	Yes				
Text output				Yes	Yes	Yes
"Quick and dirty" static SQL analysis					Yes	
Static SQL supported	Yes		Yes	Yes	Yes	
Dynamic SQL supported	Yes	Yes	Yes	Yes		Yes*
CLI applications supported	Yes		Yes	Yes		
Available to DRDA Application Requesters			Yes			
Detailed optimizer information	Yes	Yes	Yes	Yes		
Suited for analysis of multiple statements			Yes	Yes	Yes	Yes

Table 16. Explain Facility Tools (continued)

Desired Characteristics	Visual Explain	db2vexp	Explain tables	db2exfmt	db2expln	dynexpln
Information accessible from within an application			Yes			
Note: * Indirectly using db2expln; there are some limitations.						

Using the SQL Explain Facility

The different means of capturing explain information include using:

1. EXPLAIN and EXPLSNAP BIND/PREP options
2. CURRENT EXPLAIN MODE and CURRENT EXPLAIN SNAPSHOT special registers
3. EXPLAIN SQL statement
4. *db2vexp* tool (also directly calls Visual Explain to display the information)

There are three reasons you may wish to collect and use explain data:

1. To understand the steps (the access plan) that the Database Manager must perform to satisfy your query. "Data Access Concepts and Optimization" on page 153 provides information which you may need to reference if you wish to understand the explain output.
2. To help evaluate your performance tuning initiatives. There are a number of actions you can take to help improve the performance of your queries. Many of these possible actions are described in subtopics of the following:
   - "Chapter 3. Application Considerations" on page 41
   - "Chapter 4. Environmental Considerations" on page 87
   - "Chapter 5. System Catalog Statistics" on page 107.

   After making a change in any of these areas, you can use the SQL explain facility to determine the impact, if any, that the change has on the access plan chosen. For example, if you add an index based on the recommendations provided in "Indexing Impact on Query Optimization" on page 93, the explain data can help you determine whether the index is, in fact, being used as you expected.

   While the explain output will provide you with information to allow you to determine the access plan that was chosen and its relative cost, the only way to accurately measure the performance improvement for a query is to use benchmark testing techniques, as described in "Chapter 12. Benchmark Testing" on page 299.

3. To help you understand the reasons for changes in query performance, you need to have the explain information both before and after your change in order to analyze the impact. Therefore, when compiling a SQL statement to the database, you should:
   - Use the explain facility to capture the plan information before your changes, and save the resulting explain tables; or, save the output from the `db2exfmt` explain tool.
   - Save and/or print the current catalog statistics if you do not want to, or cannot, access Visual Explain to view this information. (The `db2look` productivity tool, described in “Modeling Production Databases” on page 135, could be used to help perform this task.)
   - Save and/or print the data definition language (DDL) statements, including those for CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE TABLESPACE.

   The above information provides you with a *before* picture that you can use as a reference point for future analysis. For dynamic SQL statements, you can also collect this information when you run your application for the first time. For static SQL statements, you can also collect this information at bind time.

   When you wish to analyze the reason for a performance change, you can compare the *before* data to information you collect about the query and environment when you are starting your analysis (the *after* data).

   As a simple example, your analysis could show that an index is no longer being used as part of the access path. Using the catalog statistics information in Visual Explain, you might notice that the number of index levels (NLEVELS column) is now substantially higher than when the query was first bound to the database. You might then choose to:
   - Reorganize the index
   - Collect new statistics for your table and indexes
   - Gather explain information when rebinding your query.

   Following these actions, you might notice that the index is once again being used in the access plan and that performance of the query is no longer a problem.

Introductory Concepts for Explain

You can use explain information to analyze the access plan that the optimizer has chosen based on the choices described in “Data Access Concepts and Optimization” on page 153. For example, explain information may indicate that an index scan (see “Index Scan Concepts” on page 153) was chosen by the optimizer. In addition, it can also allow you to determine the following:

- How many index columns are used as search criteria, as described in "Range Delimiting and Index SARGable Predicates" on page 163
- Whether index-only access is used, as described in "Index-Only Access" on page 158
- Whether list prefetch will be used to read the pages, as described in "Understanding List Prefetching" on page 243.

As another example, the explain information could also help you understand how two tables are joined:

- The join method
- The order in which the tables are joined
- The occurrence and type of sorts.

Although you can use explain for SELECT, SELECT INTO, UPDATE, INSERT, VALUES, VALUES INTO, and DELETE SQL statements, the primary use of explain is to observe the access paths for the SELECT parts of your statements.

To satisfy an SQL query, the Database Manager typically:

- Uses one or more data objects (a table, an index, or both)
- Performs one or more operations (for example, table scan, index scan, and join)
- Returns the result set to the calling application.

For a simple SQL query, such as:

```
SELECT DEPTNO, DEPTNAME
  FROM DEPARTMENT
```

the following, graphical representation of the steps performed could be displayed by Visual Explain:

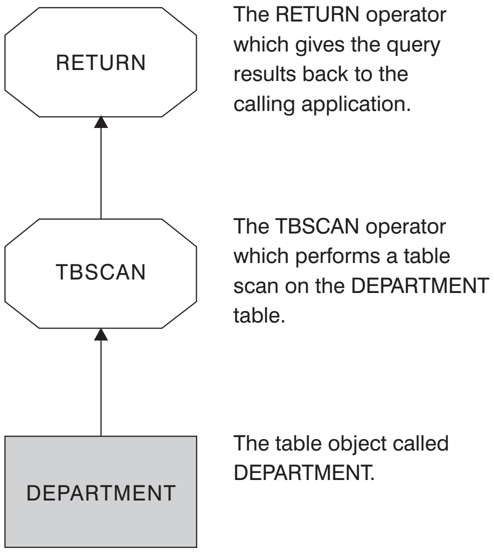

RETURN
The RETURN operator which gives the query results back to the calling application.
TBSCAN
The TBSCAN operator which performs a table scan on the DEPARTMENT table.
DEPARTMENT
The table object called DEPARTMENT.

Figure 20. Graphical Display of Explain Output

The following topics discuss the type of details you can view for objects and operators:

- "Explain Information for Data Objects"
- "Explain Information for Data Operators" on page 208

Explain Information for Data Objects

A single access plan may use one or more data objects to satisfy the SQL statement.

Object Statistics: The explain facility records facts about the object, such as:

- The creation time
- The last time that statistics were collected for the object (see "Chapter 5. System Catalog Statistics" on page 107)
- An indication of whether or not the data in the object is ordered
- The number of columns in the object
- The estimated number of rows in the object
- The number of pages that the object occupies in the buffer pool
- The total estimated overhead, in milliseconds, for a single random I/O to the specified table space where this object is stored
- The estimated transfer rate, in milliseconds, to read a 4K page from the specified table space
- Prefetch and extent sizes, in 4K pages
- The degree of data clustering with the index

- The number of leaf pages used by this object's index and the number of levels in the tree
- The number of distinct full key values in this object's index
- The total number of overflow records in the table.

Explain Information for Data Operators

A single access plan may perform several operations on the data to satisfy the SQL statement and provide results back to you. The SQL compiler determines the operations required; for example, a table scan, an index scan, a nested loop join, or a group-by. Details of many of these operators are provided in "Data Access Concepts and Optimization" on page 153.

In addition to showing the various operators used in an access plan, explain information is also available for each operator as well as the cumulative effects of the access plan.

Estimated Cost Information: The following estimated, cumulative costs can be displayed for the operators. These costs are for the chosen access plan, up to and including the operator for which the information is captured.

- The total cost (in timerons)
- The number of page I/Os
- The number of CPU instructions
- The cost (in timerons) of fetching the first row, including any initial overhead required
- The communication cost (in frames).

Timerons are a made-up, relative unit of measure.

Operator Properties: The following information is recorded by the explain facility to describe the properties of each operator:

- The set of tables that have been accessed
- The set of columns that have been accessed
- The columns on which the data is ordered, if the optimizer determined that this ordering can be used by subsequent operators
- The set of predicates that have been applied
- The estimated number of rows that will be returned (cardinality).

How Explain Information is Organized

All explain information is organized around the concept of an explain instance. An explain instance represents one invocation of the explain facility for one or more SQL statements. An explain instance represents the explain information for:

- All the eligible SQL statements in one package for **static** SQL statements
- One particular SQL statement for incremental bind SQL statements
- One particular SQL statement for **dynamic** SQL statements
- Each EXPLAIN SQL statement (whether dynamic or static).

The explain information captured within one explain instance includes the SQL Compilation environment as well as the access plan chosen to satisfy the SQL statement being compiled. Explain information is organized into 3 subsets:

Explain Instance Information	Compilation environment information captured for each explain instance.
Explain Snapshot Information	Information used by Visual Explain.
Explain Table Information	Information collected when explain table information is requested.

Explain Instance Information

Explain instance information is stored in the EXPLAIN_INSTANCE table. Additional specific information about each SQL statement explained within an explain instance is stored in the EXPLAIN_STATEMENT table.

Explain Instance Identification: You can uniquely identify each explain instance and correlate the information for the SQL statements to a given invocation of the facility with this information:

- The user who requested the explain information
- When the explain request began
- The name of the package from which the explained SQL statement came
- The schema of the package from which the explained SQL statement came.
- An indication whether a snapshot was part of the explain request.

Environmental Settings: Environmental information concerning how the SQL compiler optimized your queries is captured. The environmental information includes the following:

- The version and release number for the level of DB2 being used.
- The degree of parallelism used to compile the query. The CURRENT DEGREE special register, the DEGREE bind option, the SET RUNTIME DEGREE API, and the *dft_degree* configuration parameter may be used to determine the degree of parallelism to be used when compiling a particular query.
- Whether the SQL statement was dynamic or static.
- The query optimization class used to compile the query. See “Adjusting the Optimization Class” on page 64 for more information.

- The type of cursor blocking specified when compiling the query. For more information about cursors, refer to the *SQL Reference* manual. For more information about cursor blocking, see “Row Blocking” on page 76.
- The isolation level used when compiling the query. See “Concurrency” on page 41 for more information.
- The values of various configuration parameters when the query was compiled. See “Configuration Parameters Affecting Query Optimization” on page 87 for more information about the configuration parameters that can affect query optimization, including the following parameters that are recorded when an explain snapshot is taken:
  - “Buffer Pool Size (buffpage)” on page 327
  - “Sort Heap Size (sortheap)” on page 342
  - “Average Number of Active Applications (avg_appls)” on page 375
  - “Database Heap (dbheap)” on page 329
  - “Maximum Storage for Lock List (locklist)” on page 335
  - “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 364
  - “CPU Speed (cpuspeed)” on page 451
  - “Communications Bandwidth (comm_bandwidth)” on page 450.

SQL Statement Identification: For each explain instance, multiple SQL statements may have been explained. Along with information that uniquely identifies the explain instance, the following information helps identify each individual SQL statement.

- The type of statement: SELECT, DELETE, INSERT, UPDATE, positioned DELETE, positioned UPDATE.
- The statement and section number of the package issuing the SQL statement, as recorded in SYSCAT.STATEMENTS catalog view.

Within the EXPLAIN_STATEMENT table, the QUERYTAG and QUERYNO fields contain identifiers and are set for you as part of the explain process.

For dynamic explain SQL statements submitted during a CLP or CLI session, when EXPLAIN MODE or EXPLAIN SNAPSHOT is active, the QUERYTAG is set to “CLP” or “CLI”. When this happens, the QUERYNO is defaulted to a number that is incremented by one or more for each statement.

For all other dynamic explain SQL statements (not from CLP, CLI, or using the EXPLAIN SQL statement) the QUERYTAG is set to blanks, and the QUERYNO will always be “1”.

Cost Estimation: For each statement explained, an estimate of the relative cost of executing the chosen access plan is recorded. This cost is given using a made-up, relative unit of measure called *timerons*. Estimates of elapsed times are **not** provided, for the following reasons:

- The SQL optimizer does not estimate elapsed time but rather resource consumption.
- The optimizer does not model all factors that can affect elapsed time; it ignores those that do not affect the efficiency of the access plan. The elapsed time **is** affected by a number of run-time factors including: the system workload; the amount of resource contention; the amount of parallel processing and I/O; the cost of returning rows to the user; and the communication time between the client and server.

Statement Text: For each statement explained, two versions of the text of the SQL statement are recorded. One version is the text as received by the SQL Compiler. The other is a version of the statement text that has been reverse-translated from the internal compiler representation of the query. This translation, while looking similar to other SQL statements, does **not** necessarily follow correct SQL syntax nor does it necessarily reflect the actual content of the internal representation as a whole. This translation is provided simply to allow an understanding of the SQL context from which the SQL optimizer chose the access plan. Comparing the user-written statement text to the internal representation of the SQL statement can help you to understand how the SQL compiler has rewritten your query for better optimization. (See "Rewrite Query by the SQL Compiler" on page 143.) It also shows you other elements in the environment affecting your statement such as triggers and constraints. Some keywords used by this "optimized" text are:

$Cn	The name of a derived column, where n represents an integer value.
$CONSTRAINT$	The tag used to indicate the name of a constraint added to the original SQL statement during compilation. Seen in conjunction with the $WITH_CONTEXT$ prefix.
$DERIVED.Tn	The name of a derived table, where n represents an integer value.
$INTERNAL_FUNC$	The tag used to indicate the presence of a function used by the SQL Compiler for the explained query but not available for general use.
$INTERNAL_PRED$	The tag used to indicate the presence of a predicate added by the SQL Compiler during compilation of the explained query. Again, such a predicate is not available for general use. An internal predicate is used by the compiler to satisfy additional context added to

	the original SQL statement as the result of triggers and constraints.
RID	The tag used to identify the Row Identifier (RID) column for a particular row.
$TRIGGER$	The tag used to indicate the name of a trigger added to the original SQL statement during compilation. Seen in conjunction with the $WITH_CONTEXT$ prefix.
$WITH_CONTEXT$(...)	This prefix will appear at the start of the text when additional triggers or constraints have been added into the original SQL statement. Following this prefix will appear a list of the names of any triggers or constraints affecting the compilation and resolution of the SQL statement.

Explain Snapshot Information

When an explain snapshot is requested, additional explain information is recorded describing the access plan selected by the SQL optimizer. This information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT table in the format required by Visual Explain. This format is not usable by other applications.

Additional information on the contents of the explain snapshot information is available from Visual Explain itself and in:
- "Explain Information for Data Objects" on page 207
- "Explain Information for Data Operators" on page 208.

Explain Table Information

When explain table information is requested, additional information is recorded describing the access plan selected by the SQL optimizer. This information is stored in the following explain tables:
- EXPLAIN_ARGUMENT. This table represents the unique characteristics for each individual operator, if any.
- EXPLAIN_INSTANCE. This table is the main control table for all Explain information. Each row of data in the Explain tables is explicitly linked to one unique row in this table. Basic information about the source of the SQL statements being explained and environment information is kept in this table.
- EXPLAIN_OBJECT. This table identifies those data objects required by the access plan generated to satisfy the SQL statement.
- EXPLAIN_OPERATOR. This table contains all the operators needed to satisfy the SQL statement by the SQL compiler.

- EXPLAIN_PREDICATE. This table identifies which predicates are applied by a specific operator.
- EXPLAIN_STATEMENT. This table contains the text of the SQL statement as it exists for the different levels of Explain information. The original SQL statement as entered by the user is stored in this table along with the version used (by the optimizer) to choose an access plan to satisfy the SQL statement.
- EXPLAIN_STREAM. This table represents the input and output data streams between individual operators and data objects. The data objects themselves are represented in the EXPLAIN_OBJECT table. The operators involved in a data stream are represented in the EXPLAIN_OPERATOR table.
- ADVISE_WORKLOAD. This table allows users to describe their workload to the database. Each row in the workload represents a SQL statement, and is described by an associated frequency. This table is used by the `db2advis` tool and the Index wizard, to pick up and store work and information.
- ADVISE_INDEX. This table stores information about recommended indexes. The table is populated by the SQL compiler, the `db2advis` utility, the Index wizard, or a user. This table is used in two ways:
  - To get recommended indexes.
  - To evaluate indexes based on input about proposed indexes.

All of the tables above are not created by default. They can be created by running the EXPLAIN.DDL script found in the `misc` subdirectory of the `sqllib` subdirectory. Connect to the database where the Explain and Advise tables are required. Then issue the command: `db2 -tf EXPLAIN.DDL` and the tables will be created. The tables could also be automatically created by the Index wizard, if necessary.

Each rectangular *object* node of Visual Explain corresponds to a row in the EXPLAIN_OBJECT table. Each octagonal "operator" node of Visual Explain corresponds to a row in the EXPLAIN_OPERATOR table. Each link between operators or operator's objects corresponds to a row of the EXPLAIN_STREAM table.

The explain table information is similar in content to that recorded for an explain snapshot, however, this information is stored in ordinary relational tables which can be accessed using standard SQL statements.

Like the Visual Explain access plan graph, explain tables are designed to reflect the relationships between operators and data objects within the access plan. The following diagram shows the relationships between these tables.

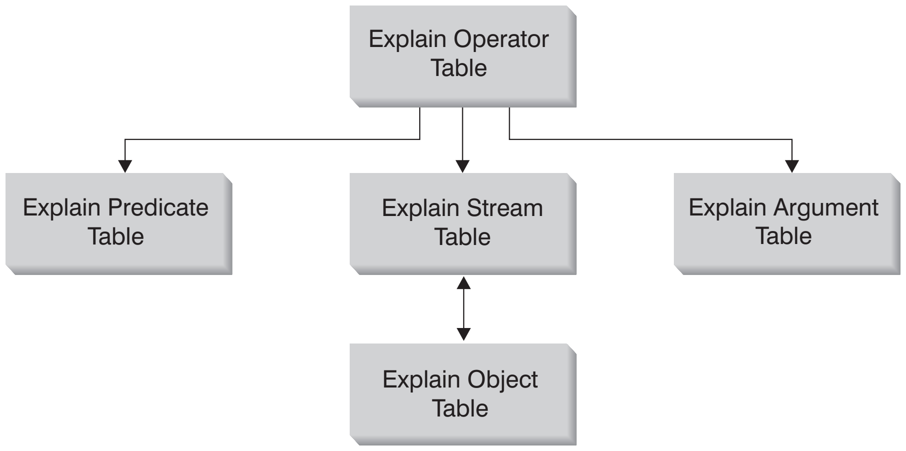


Figure 21. Overview of Explain Table Relationships (not all tables are shown).

It is possible to have explain tables that are common to more than one user. The explain tables can be defined for one user. Aliases can then be defined using the same name for each additional user pointing to the defined tables. Each user sharing the common explain tables must have insert permission on those tables.

See “Appendix C. SQL Explain Tools” on page 531 for more information on the Explain tables and how to create the tables. Additional information on the contents of the explain table information is available in:

- “Explain Information for Data Objects” on page 207
- “Explain Information for Data Operators” on page 208.

The `db2exfmt` tool provided in the `misc` subdirectory under the `sqllib` directory can be used to format the contents of the explain tables into a legible, organized output.

Obtaining Explain Data

Before you can obtain explain data for an SQL statement, you must have a set of explain tables defined using the same schema as the authorization ID that invokes the explain facility. See “Table Definitions for Explain Tables” on page 518 for information on how to create the tables.

Capturing Explain Table Information

Once these tables are defined, explain data is captured when an SQL statement is compiled and explain data has been requested:

- For **static** or **incremental bind** SQL statements, explain table information is captured when either EXPLAIN ALL or EXPLAIN YES options are specified on the BIND or the PREP commands; or, a static EXPLAIN SQL statement is used in the source program.

  Note: When incremental bind SQL statements are compiled at run-time, they are placed in the explain tables at run-time and not bind-time. Also, the explain table qualifier and authorization ID used for the insert to the explain tables is that of the package owner and not that of the use running the package.
- For **dynamic** SQL statements, explain table information is captured for any of the following situations:
  - An EXPLAIN SQL statement. All explain information is captured and placed in the explain tables unless the FOR SNAPSHOT clause is used.

    An example of an EXPLAIN SQL statement:
    ```
    EXPLAIN PLAN FOR <any valid DELETE, INSERT, SELECT, SELECT INTO,
       UPDATE, VALUES, or VALUES INTO SQL statement>
    ```
  - The CURRENT EXPLAIN MODE special register is set to YES. This setting causes the SQL compiler to capture explain data and allow the SQL statement to execute, returning the results of the query.
  - The CURRENT EXPLAIN MODE special register is set to EXPLAIN. This setting causes the SQL compiler to capture explain data, but does not execute the SQL statement.
  - The CURRENT EXPLAIN MODE special register is set to RECOMMEND INDEXES. This setting causes the SQL compiler to capture explain data and the recommended indexes to be placed in the ADVISE_INDEX table; however, the SQL statement is not executed.
  - The CURRENT EXPLAIN MODE special register is set to EVALUATE INDEXES. This setting causes the SQL compiler to use indexes placed by the user in the ADVISE_INDEX table. The user inserts a new row for each index that should be evaluated. The required information for each index is: index name, table name, and the columns names that make up the index being evaluated. Once entered, the special register, CURRENT EXPLAIN MODE should be set to EVALUATE INDEXES. Then the SQL compiler scans the ADVISE_INDEX table where the field USE_INDEX is set to “Y” (these are called virtual indexes). All dynamic statements executed in EVALUATE INDEXES mode are explained as if these virtual indexes were available. The SQL compiler then chooses to use the virtual indexes if they improve the performance of the statements. Otherwise, the indexes are ignored. By reviewing the EXPLAIN results, you can see if the indexes proposed by the user were used by the SQL compiler. Those that were used should be considered to be implemented to improve access.

– The EXPLAIN ALL option has been specified on the BIND or PREP command. This setting causes the SQL compiler to capture explain data for dynamic SQL at run-time, even if the setting of the CURRENT EXPLAIN MODE special register is NO. The SQL statement also executes, returning the results of the query.

Note: Explain information is only captured when the SQL statement is compiled. Following the initial compilation, dynamic SQL statements are only recompiled when a change to the environment requires the statement be recompiled. If the same PREPARE statement is issued consecutively for the same SQL statement, the SQL statement will only be compiled, and explain data captured, the first time the PREPARE statement is issued, assuming the environment does not change.

For more information about using the EXPLAIN SQL statement or about using the CURRENT EXPLAIN MODE registers, refer to the *SQL Reference* manual. For more information about the BIND and PREP commands, refer to the *Command Reference* manual.

Capturing Explain Snapshot Information

Explain snapshot data is captured when an SQL statement is compiled and explain data has been requested:

- For **static** or **incremental bind** SQL statements, explain snapshot is captured when either EXPLSNAP ALL or EXPLSNAP YES clauses are specified on the BIND or the PREP commands; or, a static EXPLAIN SQL statement that uses a FOR SNAPSHOT or a WITH SNAPSHOT clause is used in the source program.

  Note: When incremental bind SQL statements are compiled at run-time, they are placed in the explain tables at run-time and not bind-time. Also, the explain table qualifier and authorization ID used for the insert to the explain tables is that of the package owner and not that of the use running the package.
- For **dynamic** SQL statements, an explain snapshot is captured in any of the following situations:
  – An EXPLAIN SQL statement using a FOR SNAPSHOT or a WITH SNAPSHOT clause. The FOR SNAPSHOT clause has no explain table information captured except the information associated with explain snapshot. The WITH SNAPSHOT clause has all explain table information captured in addition to the information associated with explain snapshot.

  An example of an explain snapshot using the EXPLAIN SQL statement:

```
EXPLAIN PLAN FOR SNAPSHOT FOR <any valid DELETE, INSERT, SELECT,
   SELECT INTO, UPDATE, VALUES, or VALUES INTO SQL statement>
```

Only an explain snapshot is taken and the captured information is placed in the EXPLAIN_INSTANCE and EXPLAIN_STATEMENT tables.
  - The CURRENT EXPLAIN SNAPSHOT special register is set to YES. This setting causes the SQL compiler to take a snapshot of explain data and allows the SQL statement to execute, returning the results of the query.
  - The CURRENT EXPLAIN SNAPSHOT special register is set to EXPLAIN. This setting causes the SQL compiler to take a snapshot of explain data, but does not execute the SQL statement.
  - The EXPLSNAP ALL option has been specified on the BIND or PREP command. This setting causes the SQL compiler to take a snapshot of explain data at run-time, even if the setting of the CURRENT EXPLAIN SNAPSHOT special register is NO. The SQL statement will also execute, returning the results of the query.

Note: Explain information is only captured when the SQL statement is compiled. Following the initial compilation, dynamic SQL statements are only recompiled when a change to the environment requires the statement be recompiled. If the same PREPARE statement is issued consecutively for the same SQL statement, the SQL statement will only be compiled, and explain data captured, the first time the PREPARE statement is issued, assuming the environment does not change.

For more information about using the EXPLAIN SQL statement and the FOR SNAPSHOT or WITH SNAPSHOT clauses, or about using the CURRENT EXPLAIN SNAPSHOT registers, refer to the *SQL Reference* manual. For more information about the BIND and PREP commands, refer to the *Command Reference* manual.

Guidelines on Using Explain Output

There are a number of ways in which analyzing the explain data can help you to tune your queries and environment. For example:

- **Are Indexes Being Used?**

  As discussed in "Indexing Impact on Query Optimization" on page 93, the proper indexes can have a significant benefit on performance. Using the explain output, you can determine if the indexes you have created to help a specific set of queries are being used. In the explain output, you should look for index usage in the following areas:
  - Join predicates
  - Local predicates
  - GROUP BY clause
  - ORDER BY clause

  – The select list.

  You can also use the explain facility to evaluate whether a different index can be used instead of an existing index, or no index at all. After creating a new index, collect statistics for that index (using the RUNSTATS command) and recompile your query. Over time you may notice through the explain data that instead of an index scan, a table scan is now being used. This can result from a change in the clustering of the table data. If the index that was previously being used now has a low cluster ratio, you may want to:

  – Reorganize your table to cluster the data according to that index
  – Use the RUNSTATS command to update the catalog statistics for the table and index
  – Recompile your query
  – Re-examine the explain output to determine whether reorganizing your table has impacted the access plan.

- **Is the Type of Access Appropriate for Your Application?**

  You can analyze the explain output and look for types of access to the data that, as a rule, are not optimal for the type of application you are running. For example:

  – **Online Transaction Processing (OLTP) Queries**

    OLTP applications are prime candidates to use index scans with range delimiting predicates, because they tend to return only a few rows that are qualified using an equality predicate against a key column. If your OLTP queries are using a table scan, you may want to analyze the explain data to determine the reasons why an index scan was not used.

  – **Browse-Only Queries**

    The search criteria for a "browse" type query may be very vague, causing a large number of rows to qualify. If the user will usually only look at a few screens of the output data, you may want to try to ensure that the entire answer set need not be computed before some results are returned. In this case, the goals of the user are different from the basic operating principle of the optimizer, which attempts to minimize resource consumption for the entire query, not just the first few screens of data.

    For example, if the explain output shows that both merge scan join and sort operators were used in the access plan, then the entire answer set will be materialized in a temporary table before any rows are returned to the application. In this case, you can attempt to change the access plan by using the OPTIMIZE FOR clause on the SELECT statement. (For more information on the OPTIMIZE FOR clause, see "OPTIMIZE FOR n ROWS Clause" on page 73.) In this way, the optimizer can attempt to choose an access plan that does not produce the entire answer set in a temporary table before returning the first rows to the application.

- **What Type of Join Method is Being Used?**

  If a query joins two tables, you can check the type of join processing being used. Joins involving more rows, such as those in decision-support queries, usually run faster with a merge join. Joins involving only a few rows, such as OLTP queries, typically run faster with nested loop joins. However, there may be extenuating circumstances in either case, such as the use of local predicates or indexes, that would change how these typical joins would work. (See "Nested Loop Join" on page 165 and "Merge Join" on page 167 for information about how these two join methods operate.)

Visual Explain

Visual Explain can be used to study queries in more detail when compared to the other methods, especially those that contain more complex sequences of operations. Visual Explain is not available on all supported platforms. You should check the *Quick Beginnings* for your platform to see if Visual Explain is supported.

Visual Explain lets you view the access plan for explained SQL statements as a graph. You can use the information available from the graph to tune your SQL queries for better performance. Visual Explain also lets you dynamically explain a SQL statement and view the resulting access plan graph.

The optimizer chooses an access plan and Visual Explain displays the information as an access plan graph in which tables and indexes, and each operation on them, are represented as nodes, and the flow of data is represented by the links between the nodes.

To display an access plan graph, you must have created an explain snapshot. From an access plan graph, you can view the details for:

- Tables and indexes (and their associated columns)
- Operators (such as table scans, sorts, and joins)
- Table spaces and functions.

You can also use Visual Explain to:

- View the statistics that were used at the time of optimization. You can then compare these statistics to the current catalog statistics to help you determine whether rebinding the package might improve performance.
- Determine whether or not an index was used to access a table. If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- View the effects of performing various tuning techniques by comparing the before and after versions of the access plan graph for a query.

- Obtain information about each operation in the access plan, including the total estimated cost and number of rows retrieved (cardinality).

For additional detail on Visual Explain, you should refer to the online information available through the Control Center. The Control Center can be accessed by typing `db2cc` on the command line.

SQL Advise Facility

The Index Advisor is a management tool that reduces the need for you to design and define suitable indexes for your data.

The Index Advisor is good for:
- Finding the best indexes for a problem query.
- Finding the best indexes for a set of queries (a workload), subject to resource limits which are optionally applied.
- Testing an index on a workload without having to create the index.

There are concepts associated with the SQL Advise Facility. First, there is a *workload*. A workload is a set of SQL statements which the database manager has to process over a given period of time. The SQL statements can include: SELECT, INSERT, UPDATE, and DELETE statements. For example, over a one month period of time your database manager may have to process 1 000 INSERTs, 10 000 UPDATEs, 10 000 SELECTs, and 1 000 DELETEs. The information in the workload is concerned with the type and frequency of the SQL statements over a given period of time. The advising engine uses this workload information in conjunction with the database information to recommend indexes. The goal of the advising engine is to minimize the total workload cost.

Second, there is a concept of a *virtual index*. Virtual indexes are indexes which do not exist in the current database schema. These indexes could be either recommendations that the Advise Facility has made to you, or indexes that you are looking to the Advise Facility to evaluate for you. These indexes could also be those the Advise Facility considers as part of the process and then discards because they are not going to be recommended. Virtual indexes are passed back and forth from you to the Advise Facility using the ADVISE_INDEX table.

The Advise Facility uses a workload and statistics from the database to generate recommended indexes.

The Advise Facility uses two EXPLAIN tables:
- ADVISE_WORKLOAD

This table is where you describe the workload to be considered. Each row in the table represents an SQL statement and is described by an associated frequency. There is an identifier for each workload that is a field of the table called "WORKLOAD_NAME". All SQL statements which are part of the same workload should have the same WORKLOAD_NAME.

The Index wizard and the `db2advis` tool use the table to pick up and store workload information.

- ADVISE_INDEX

  This table stores information about recommended indexes. Information is placed into this table from the SQL compiler, the Index wizard, the `db2advis` tool, or you.

  The table is used in two ways:
  - To get recommended indexes from the Advise Facility
  - To evaluate indexes.

Note: To create these table, run the EXPLAIN.DDL script found in the `misc` subdirectory of the `sqllib` subdirectory. If not already created, the Index wizard can also create these table.

The process for using the Index Advisor involves inputs, invocation of the advisor, outputs, and some special cases that should be considered.

There are three ways to create the input for the Index Advisor:
- Capturing a workload.

  That is, using one of the following ways to create the SQL to be evaluated:
  - Using the monitor to get dynamic SQL.
  - Using the SYSSTMT catalog view to get static SQL.
  - Adding statements and frequencies by cutting and pasting the values into the ADVISE_INDEX table.
- Modifying the workload frequencies to increase or decrease the importance of queries.
- Determining the constraints, if any, on the data.

There are four ways to invoke the Index Advisor:
- Using the Control Center.

  This is the recommended way to use the Index Advisor. From the Control Center, expand the object tree until you find the **indexes** folder. Click with mouse button two on the **indexes** folder and select **Create–>Index using wizard** from the pop-up menu. The Index wizard opens. There is extensive help with the Index wizard and it is easy to use. The wizard also contains features to construct a workload by looking for recently executed SQL, or looking through the recently used packages, or by manually adding SQL statements.

- Using the command line processor.

  On the command line enter `db2advis`. The `db2advis` starts by reading in a workload from one of three locations:

  – From the command line

  – From the statements in a text file

  – From the ADVISE_WORKLOAD table after you have inserted rows with the proposed workload (SQL and frequency).

  The tool then uses the CURRENT EXPLAIN MODE register to obtain recommended indexes, combined with an internal optimization algorithm for picking out the best indexes. The output goes to your terminal screen, the ADVISE_INDEX table, and an output file, if desired.

  For example, you may wish the tool to recommend indexes for a simple query “select count(*) from sales where region = ′Quebec′”

```
$ db2advis -d sample \
 -s "select count(*) from sales where region = 'Quebec'" \
 -t 1
performing auto-bind

Bind is successful. Used bindfile: /home3/valentin/sqllib/bnd/db2advis.bnd

Calculating initial cost (without recommended indexes) [31.198040] timerons
Initial set of proposed indexes is ready.
Found maximum set of [1] recommended indexes
Cost of workload with all indexes included [2.177133] timerons
cost without index [0] is [31.198040] timerons.  Derived benefit is
[29.020907]
total disk space needed for initial set [1] MB
total disk space constrained to         [-1] MB
  1  indexes in current solution
 [31.198040] timerons  (without indexes)
 [2.177133] timerons  (with current solution)
 [%93.02] improvement

 Trying variations of the solution set.
Time elapsed.
LIST OF RECOMMENDED INDEXES
===========================
index[1], 1MB CREATE INDEX WIZ689 ON VALENTIN.SALES (REGION DESC)
===========================
Index Advisor tool is finished.
```

  The `db2advis` tool can be used to recommend indexes for a workload as well. You can create an input file called “sample.sql”:

```
--#SET FREQUENCY 100
select count(*) from sales where region = ?;
--#SET FREQUENCY 3
select projno, sum(comm) tot_comm  from employee, emp_act
where employee.empno = emp_act.empno and
```

```
     employee.job='DESIGNER'
group by projno
order by tot_comm desc;
--#SET FREQUENCY 50
select * from sales where sales_date = ?;
```

Then execute the following command:

```
$ db2advis -d sample -i sample.sql -t 0
  found [3] SQL statements from the input file

Calculating initial cost (without recommmended indexes) [62.331280] timerons
Initial set of proposed indexes is ready.
Found maximum set of [2] recommended indexes
Cost of workload with all indexes included [29.795755] timerons
cost without index [0] is [58.816662] timerons.  Derived benefit is
[29.020907]
cost without index [1] is [33.310373] timerons.  Derived benefit is
[3.514618]
total disk space needed for initial set [2] MB
total disk space constrained to         [-1] MB
  2  indexes in current solution
 [62.331280] timerons  (without indexes)
 [29.795755] timerons  (with current solution)
 [%52.20] improvement

Trying variations of the solution set.
Time elapsed.
LIST OF RECOMMENDED INDEXES
===========================
index[1], 1MB CREATE INDEX WIZ119 ON VALENTIN.SALES (SALES_DATE DESC,
SALES_PERSON DESC)
index[2], 1MB CREATE INDEX WIZ63 ON VALENTIN.SALES (REGION DESC)
===========================
Index Advisor tool is finished.
```

- Using self-directed methods involving the EXPLAIN modes and PREP options.

  For example, the CURRENT EXPLAIN MODE special register is set to RECOMMEND INDEXES. This setting will cause the SQL compiler to capture explain data and the recommended indexes to be placed in the ADVISE_INDEX table; however, the SQL statement is not executed.

  Or, the CURRENT EXPLAIN MODE special register is set to EVALUATE INDEXES. This setting will cause the SQL compiler to use indexes placed by the user in the ADVISE_INDEX table. The user inserts a new row for each index that should be evaluated. The required information for each index is: index name, table name, and the columns names that make up the index being evaluated. Once entered, the special register CURRENT EXPLAIN MODE should be set to EVALUATE INDEXES. Then the SQL compiler scans the ADVISE_INDEX table for indexes where the field USE_INDEX="Y" (these are called virtual indexes). All dynamic statements executed in EVALUATE INDEXES mode are explained as if these virtual indexes were available. The SQL compiler then chooses to use the virtual indexes if they improve the performance of the statements. Otherwise, the indexes are ignored. By reviewing the EXPLAIN results, you can see if the indexes proposed by the user were used by the SQL compiler. Those that were used should be considered to be implemented to improve access.

- Using the Call Level Interface (CLI).

  If you are using this interface to write applications, you can also use the advisor.

There are different ways to use the results from the advisor:

- Interpreting the output from the Index Advisor.

  To see what indexes were recommended by the Advise Facility, you can use the following query:

  ```
  SELECT CAST(CREATION_TEXT as CHAR(200))
     FROM ADVISE_INDEX
  ```
- Applying the recommendations of the Index Advisor.
- Knowing when to drop an index.

To get better recommendations for a specific query, it is suggested that you advise that query by itself. You can use the Index wizard to recommend indexes for a single query by building a workload which contains only that query.

A sample workload can be collected from Event Monitor output. The Event Monitor can be used to collect dynamic SQL executions. Then these statements can be fed back to the Advise Facility.

The Index wizard is a simple, straight-forward, easy to use, visual interface providing an excellent way to access the Advise Facility.

Part 3. Tuning and Configuring Your System

Chapter 8. Operational Performance

The following topics provide information on how you can influence performance of an SQL query during run-time:

- How DB2 Uses Memory
- Managing the Database Buffer Pool
- Managing Multiple Database Buffer Pools
- Prefetching Data into the Buffer Pool
- Configuring I/O Servers for Prefetching and Parallel I/O
- Sorting
- Reorganizing Catalogs and User Tables
- Performance Considerations for DMS Devices
- Managing Initialization Overhead
- Database Agents
- Using the Database System Monitor
- Extending Memory.

The following chapters also provide information on how performance can be influenced:

- "Chapter 3. Application Considerations" on page 41
- "Chapter 4. Environmental Considerations" on page 87
- "Chapter 5. System Catalog Statistics" on page 107.

You may also refer to *Administration Guide: Planning* physical database design considerations.

How DB2 Uses Memory

Many of the configuration parameters available in DB2 affect memory usage on the system. Some may affect memory on the server, some on the client, and some on both. Furthermore, memory is allocated and de-allocated at different times and from different areas of the system.

A system administrator should also take into consideration balancing overall memory usage on the system. Different applications running on the operating system may use memory in different ways. For example, some applications may use the file system cache, while the Database Manager uses its own buffer pool for data caching instead of the operating system facility. See "Setting Parameters That Affect Memory Usage" on page 234 for additional considerations.

Figure 22 shows that the Database Manager uses different types of memory. There is an assumption that this figure illustrating memory use is not an Enterprise – Extended Edition, nor a multiple logical node, environment. In an Enterprise – Extended Edition or a multiple logical node environment, there are multiple Database Manager Shared Memory sets (one per node).

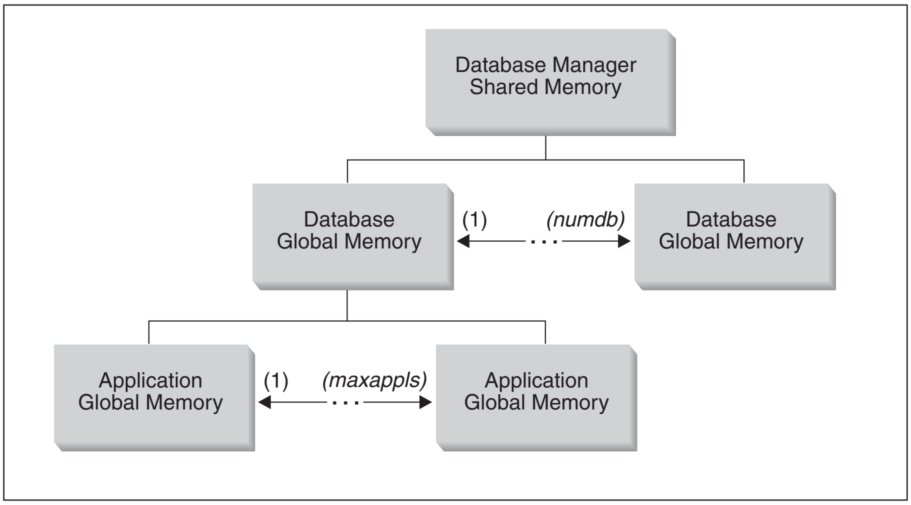

Database Manager Shared Memory
Database Global Memory
(1)
(numdb)
Database Global Memory
Application Global Memory
(1)
(maxappls)
Application Global Memory

Figure 22. Types of memory used by the Database Manager

Memory is allocated for each instance of the Database Manager at the following times:

- When the Database Manager is started (`db2start`), the area marked "Database Manager Shared Memory" is allocated, and this area remains allocated until the Database Manager is stopped (`db2stop`). This area contains information that is needed by the Database Manager to manage activity across all database connections. When the first application connects to a database, both global and private memory areas are allocated.
- When a database is activated or connected to for the first time, the "Database Global Memory" is allocated. The database global memory is used across all applications that might connect to the database and contains memory areas such as the buffer pools, lock list, database heap and utility heap.
- When an application connects to a database, the "Application Global Memory" is allocated (this occurs only in a partitioned database

environment, or if the *intra_parallel* configuration parameter is enabled). This memory is used by agents working on behalf of the application to share data and coordinate activities amongst themselves.

- (Not shown in the previous diagram:) When an agent is assigned to work for a particular application (as the result of a connect request, or, in a parallel environment, a new SQL request), “Agent Private Memory” is allocated for that agent. The agent private memory area is allocated for the agent and contains memory allocations that will be used only by this specific agent, such as the sort heap and the application heap.

  Once a database is already in use by one application, any subsequent connecting applications will only have agent private memory and application global shared memory allocated on their behalf.

Figure 22 on page 228 shows how configuration parameter settings can affect memory. In particular, the parameters in the following list can limit the amount of memory that is allocated for specific purposes. (In a partitioned database environment, this memory is required on every database partition.)

- *numdb* defines the maximum number of concurrent active databases (in use by different applications). Since each database has its own global memory area, the amount of memory that can potentially be allocated grows if the value of this parameter increases.
- *maxappls* defines the maximum number of applications that can simultaneously connect to a single database. It affects the amount of memory that can potentially be allocated for “Agent Private Memory” and “Application Global Memory” for that database. (Note that this parameter can be set differently for every database.)
- (Not shown in the previous diagram:) *maxagents* (and *max_coordagents* for parallel environments) limit the number of Database Manager agents that can exist simultaneously across all active databases within an instance. Along with *maxappls*, these parameters limit the amount of memory allocated for “Agent Private Memory” and “Application Global Memory”. (For information on agents, see “Database Agents” on page 257.)

Figure 23 on page 230 summarizes how much memory is used to support applications. The following configuration parameters allow you to control the size of this memory, by limiting the number of ″memory segments″ (portions of logical memory) and their size.

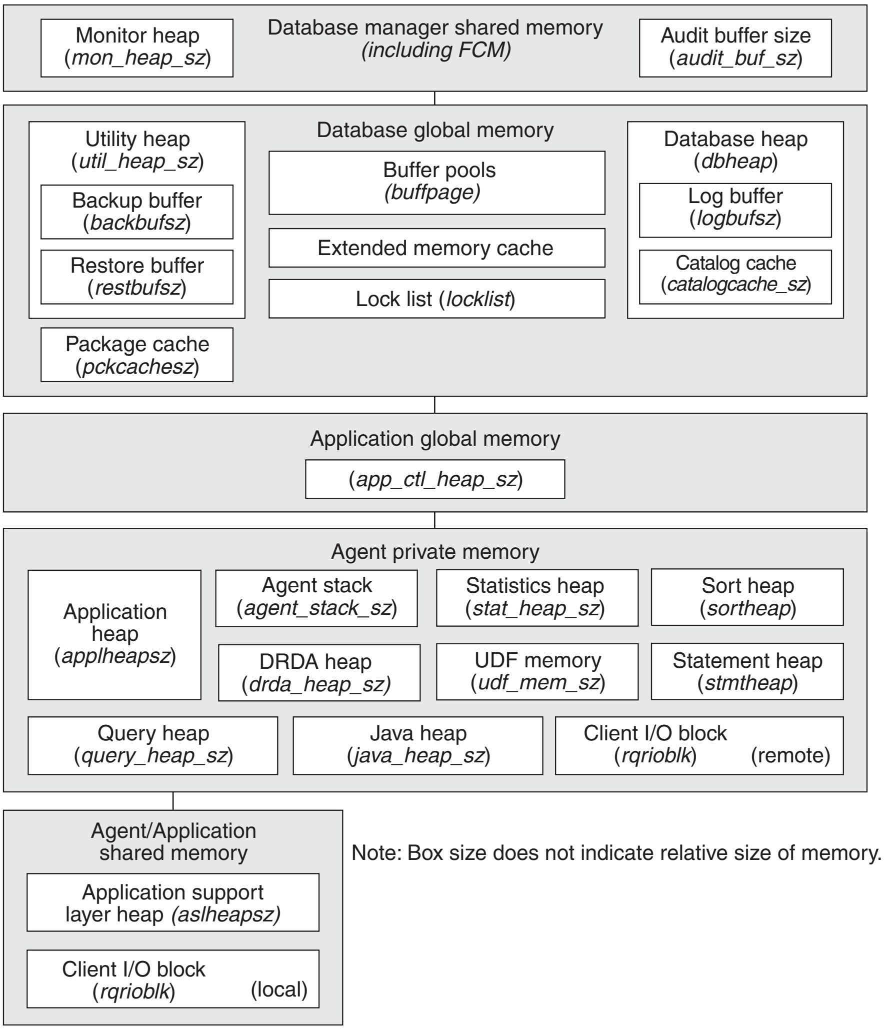


Figure 23. How Memory Is Used by the Database Manager

Database Manager Shared Memory

Memory space is required for the database manager to run. This space can be very large, especially in intra-partition and inter-partition parallelism environments. You can predict and control the size of this space by reviewing the following sections:

- "Database Agents" on page 257. Agents running on behalf of applications require substantial memory space, especially if the value of *maxagents* is not appropriate.

- “FCM Requirements” on page 235. For partitioned database systems, the fast communications manager (FCM) requires substantial memory space, especially if the value of *fcm_num_buffers* is not appropriate.

  The FCM memory requirements are either allocated from the FCM Buffer Pool, or from both the Database Manager Shared Memory and the FCM Buffer Pool, depending on whether or not the partitioned database system uses multiple logical nodes. See the following description of the FCM Buffer Pool for details.

FCM Buffer Pool

If you have a partitioned database system that does not have multiple logical nodes, the Database Manager Shared Memory and FCM Buffer Pool are as shown in Figure 24.

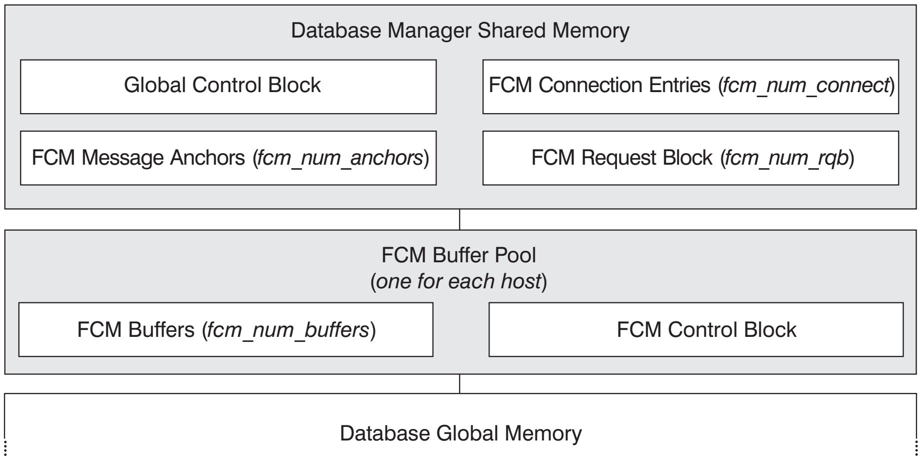
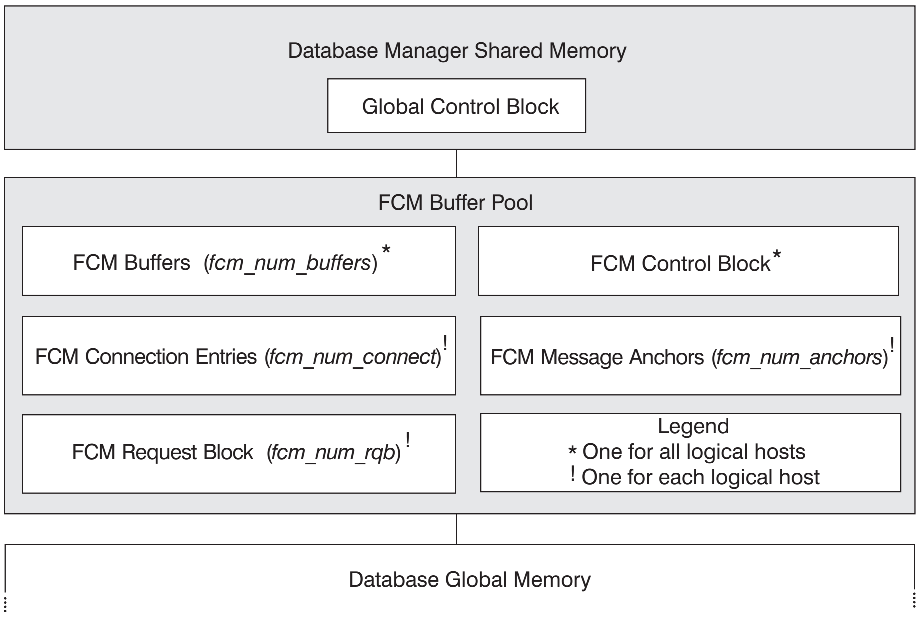

Database Manager Shared Memory
Global Control Block
FCM Connection Entries (*fcm_num_connect*)
FCM Message Anchors (*fcm_num_anchors*)
FCM Request Block (*fcm_num_rqb*)
FCM Buffer Pool
(*one for each host*)
FCM Buffers (*fcm_num_buffers*)
FCM Control Block
Database Global Memory

Figure 24. FCM Buffer Pool when Multiple Logical Nodes Are Not Used

If you have a partitioned database system that uses multiple logical nodes, the Database Manager Shared Memory and FCM Buffer Pool are as shown in Figure 25 on page 232.



Database Manager Shared Memory
Global Control Block
FCM Buffer Pool
FCM Buffers (fcm_num_buffers)*
FCM Control Block*
FCM Connection Entries (fcm_num_connect)!
FCM Message Anchors (fcm_num_anchors)!
FCM Request Block (fcm_num_rqb)!
Legend
* One for all logical hosts
! One for each logical host
Database Global Memory

Figure 25. FCM Buffer Pool when Multiple Logical Nodes Are Used

Database Global Memory

Database Global Memory is affected by the following configuration parameters:

- The number of memory segments is limited by *numdb* (see "Maximum Number of Concurrently Active Databases (numdb)" on page 451).
- The maximum size of memory segments is determined by the values of the following parameters:
  - "Buffer Pool Size (buffpage)" on page 327 (if a buffer pool size is -1), or the explicit sizes that were specified when the buffer pools were created or altered
  - "Maximum Storage for Lock List (locklist)" on page 335
  - "Database Heap (dbheap)" on page 329
  - "Utility Heap Size (util_heap_sz)" on page 333
  - "Extended Storage Memory Segment Size (estore_seg_sz)" on page 372
  - "Number of Extended Storage Memory Segments (num_estore_segs)" on page 373.
  - "Package Cache Size (pckcachesz)" on page 338.

Application Global Memory

Application Global Memory is affected by the following configuration parameter:

- “Application Control Heap Size (app_ctl_heap_sz)” on page 340.

For parallel systems, space is also required for the application control heap, which is shared between the agents that are working on behalf of the same application at one database partition. The heap is allocated when the first agent to receive a request from the application requests a connection. The agent can be either a coordinating agent or a subagent (see “Database Agents” on page 257).

Agent Private Memory

- The number of memory segments is limited by the lower of:
  - The total of *maxappls* for all active databases (see “Maximum Number of Active Applications (maxappls)” on page 374)
  - The value of *maxagents* (see “Maximum Number of Agents (maxagents)” on page 379).
- The maximum size of memory segments is determined by the values of the following parameters:
  - “Application Heap Size (applheapsz)” on page 345
  - “Sort Heap Size (sortheap)” on page 342
  - “Statement Heap Size (stmtheap)” on page 344
  - “Statistics Heap Size (stat_heap_sz)” on page 345
  - “Query Heap Size (query_heap_sz)” on page 346
  - “DRDA Heap Size (drda_heap_sz)” on page 347
  - “UDF Shared Memory Set Size (udf_mem_sz)” on page 348
  - “Agent Stack Size (agent_stack_sz)” on page 349.

Agent/Application Shared Memory

- The total number of agent/application shared memory segments (for local clients) is limited by the lower of:
  - The total of *maxappls* for all active databases (see “Maximum Number of Active Applications (maxappls)” on page 374)
  - The value of *maxagents* (see “Maximum Number of Agents (maxagents)” on page 379), or (for parallel systems) *max_coordagents* (see “Maximum Number of Coordinating Agents (max_coordagents)” on page 381).
- Agent/Application Shared Memory is also affected by the following:
  - “Application Support Layer Heap Size (aslheapsz)” on page 353.
  - “Client I/O Block Size (rqrioblk)” on page 355.

Setting Parameters That Affect Memory Usage

Parameters that allocate memory should *never* be set at their highest values, even on systems with the maximum amount of memory installed, unless such a value has been carefully justified. Many of the parameters can allow the Database Manager to very easily and quickly take up all of the available memory on a machine. In addition, the management of a large amount of memory can take significant additional work on the part of the Database Manager and thus incur even more overhead.

Some UNIX-based operating systems allocate swap space when a process allocates memory and not when it is paged out to swap space. In these cases, you should ensure the total shared memory size is backed with the equivalent amount of paging space.

For most of the configuration parameters, memory is only committed as it is required. These parameters reflect the maximum size of a particular memory heap. The notable exceptions to this rule are the following parameters for which memory is fully committed based on the parameter value:

- "Buffer Pool Size (buffpage)" on page 327 (if a buffer pool size is -1), or the explicit sizes that were specified when the buffer pools were created or altered
- "Sort Heap Threshold (sheapthres)" on page 342
- "Maximum Storage for Lock List (locklist)" on page 335
- "Application Support Layer Heap Size (aslheapsz)" on page 353
- "Number of FCM Message Anchors (fcm_num_anchors)" on page 438
- "Number of FCM Buffers (fcm_num_buffers)" on page 439
- "Number of FCM Connection Entries (fcm_num_connect)" on page 440
- "Number of FCM Request Blocks (fcm_num_rqb)" on page 441.

The appropriate values for these types of parameters can best be determined by benchmarking, where typical and worst-case SQL statements are run against the server and the values of the parameters are modified until the point of diminishing return for performance is found. If performance versus parameter values were graphed, the point where the curve begins to plateau or decline would indicate the point at which additional allocation provides no additional value to the application and is therefore simply wasting memory. (See "Chapter 12. Benchmark Testing" on page 299.)

The upper limits of memory allocation for several parameters may be beyond the memory capabilities of existing hardware and operating systems. These limits were chosen to allow for future growth.

For valid parameter ranges, see the parameter descriptions in "Chapter 13. Configuring DB2" on page 311.

FCM Requirements

Start with default values when configuring the following Fast Communications Manager (FCM) configuration parameters:

- "Number of FCM Buffers (fcm_num_buffers)" on page 439
- "Number of FCM Request Blocks (fcm_num_rqb)" on page 441
- "Number of FCM Connection Entries (fcm_num_connect)" on page 440
- "Number of FCM Message Anchors (fcm_num_anchors)" on page 438.

To tune these parameters, use the database system monitor to monitor the low water mark for the free buffers, free message anchors, free connection entries, and the free request blocks. If the low water mark is less than 10 percent of the number of the corresponding free data item, increase the value of the corresponding parameter. For information on the database system monitor, see "Using the Database System Monitor" on page 263.

Refer to *Administration Guide: Planning* for information on enabling FCM communications.

Managing the Database Buffer Pool

A buffer pool is an area of storage into which database pages (containing table rows or index entries) are temporarily read and changed. The purpose of the buffer pool is to improve database system performance. Data can be accessed much faster from memory than from a disk. Therefore, the fewer times the Database Manager needs to read from or write to a disk, the better the performance.

The configuration of one or more buffer pools is the single most important tuning area, since it is here that most of the data manipulation takes place for applications connected to the database (excluding large objects and long field data).

When an application accesses a row of a table for the first time, the database manager places the page containing that row in the buffer pool. The next time any application requests data, the buffer pool is checked first. If the requested data is found on pages kept in the buffer pool, the database manager does not need to go out to disk storage to retrieve the requested data. Avoiding the need to retrieve data from disk storage results in faster performance.

The storage associated with the buffer pool is allocated when a database is activated or when the first application connects to the database. Applications are the primary beneficiaries of the buffer pool; once applications are all disconnected, the storage associated with the buffer pool is de-allocated.

Pages stay in the buffer pool until the database is shut down, or until the space occupied by a page is required for another page. The space chosen in the buffer pool to bring in another page is selected using criteria such as the following:
- The last reference to a page
- The likelihood of the page being referenced again by the last agent that looked at the page
- The type of page
- Whether or not a page was changed in memory but not written out to disk. (Changed pages are always written to disk before being overwritten.)

Note: After changed pages are written out to disk, they are not removed from the buffer pool unless the space they occupy is needed for other pages. Until they are overwritten, they can be accessed again if their data is needed.

When creating a buffer pool, by default the page size is 4 KB. You can choose to have the page size set at one of 4 KB, 8 KB, 16 KB, or 32 KB when creating the buffer pool. If buffer pools are created using one page size, only table spaces created using the identical page size can be associated with them. You cannot alter the page size of the buffer pool following its creation.

Pages in the buffer pool can have different attributes:
- In-use pages are currently being read or updated. They can be read, but not updated, by other agents.
- "Dirty" pages are pages where data has been changed but has not yet been written to disk. After a page is written to disk, it is considered "clean", and remains in the buffer pool. The space occupied by clean pages can be used for new pages, and is available for migration to an associated extended storage cache (if defined).

Pages can be written from the buffer pool to disk when the percentage of space occupied by changed pages in the buffer pool has exceeded the value specified by the *chngpgs_thresh* configuration parameter. You also may need to configure the database to include more than one page-cleaner agent. These agents write out changed pages to disk so that the database agents can find usable space in the buffer pool.

Page cleaner agents perform I/O that would otherwise have to be performed by the database agents. As a result, your applications can run faster, because transactions are not forced to wait while their database agents write pages to disk. (Page-cleaner agents are sometimes referred to as *asynchronous page cleaners* or *asynchronous buffer writers* because they can run in parallel with the database agents.)

To change the number of page-cleaner agents, use the *num_iocleaners* configuration parameter (the default is to create one page-cleaner agent). For information, see "Number of Asynchronous Page Cleaners (num_iocleaners)" on page 367. Set the value of this parameter to between one and the number of physical disks in the database. The larger this number, the better the performance when carrying out update-intensive workloads. This is also true when there are a large number of data- or index-page-writes with respect to the number of asynchronous data- or index-page-writes.

Writing pages to disk also allows for faster recovery of the database should a system crash occur, because the Database Manager is able to rebuild more of the buffer pool from disk rather than having to use the database log files. As a result, page cleaning is requested if the size of the log that would need to be read during recovery exceeds the following maximum:

```
logfilsiz * softmax
```

where:

- *logfilsiz* represents the size of the log files (see "Size of Log Files (logfilsiz)" on page 389)
- *softmax* represents the percentage of log files to be recovered following a database crash (see "Recovery Range and Soft Checkpoint Interval (softmax)" on page 397).

  For example, if the value of softmax is 250, then 2.5 log files will contain the changes that need to be recovered if a crash occurs.

You may use the database system monitor to help you track the number of times that page cleaning is requested to minimize log read time during recovery. For more information refer to the *pool_lsn_gap_clns (buffer pool log space cleaners triggered)* monitor element description in the *System Monitor Guide and Reference* manual.

The size of the log that would need to be read during recovery is the difference between the location of the following in the log:

- The most recently written log record
- The log record that describes the oldest change to data in the buffer pool.

The following figure illustrates how the work of managing the buffer pool can be shared between page-cleaner agents and database agents, compared to the database agents performing all of the I/O.

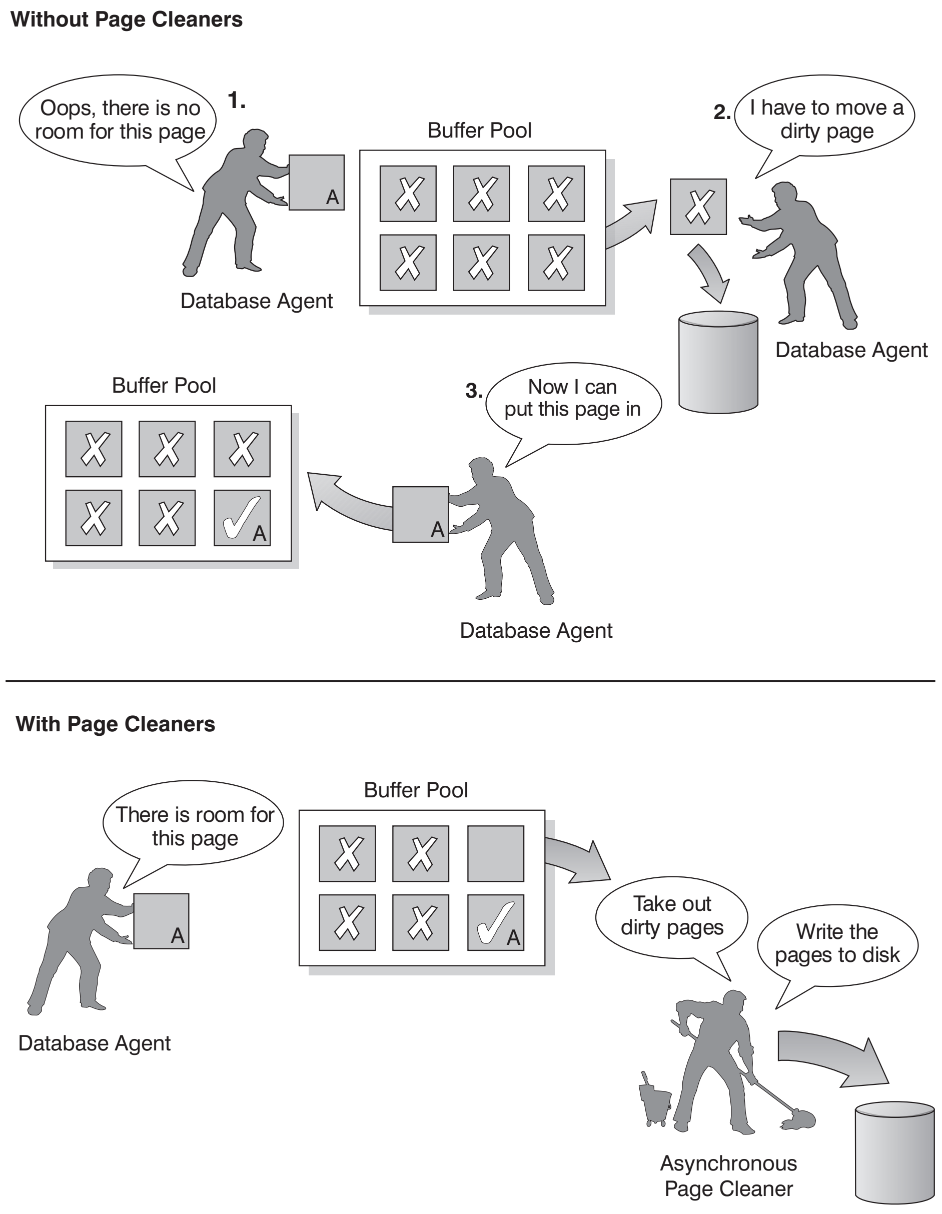


Figure 26. Asynchronous Page Cleaner. “Dirty” pages are written out to disk.

Managing Multiple Database Buffer Pools

Each database requires at least one buffer pool. However, depending on your needs you may choose to create several buffer pools, each of a different size, for a single database. The CREATE, ALTER, and DROP BUFFERPOOL statements allow you to create, change, or remove a buffer pool. You can specify which data is cached in a buffer pool with the CREATE TABLESPACE and ALTER TABLESPACE statements.

The *buffpage* configuration parameter specifies the size of any buffer pool, if the buffer pool's size is specified as -1 in the SYSCAT.BUFFERPOOLS catalog view. (Otherwise this parameter is ignored.) A buffer pool's size can be set with the DDL statements ALTER BUFFERPOOL or CREATE BUFFERPOOL.

A new database has a default buffer pool called IBMDEFAULTBP with a size determined by the platform. Once a database is created or migrated, then other buffer pools can be created for it.

When working on your database design, you may have determined that tables with 8 KB page sizes were best. As a result, you should create a buffer pool with an 8 KB page size (along with one or more table spaces with the same page size).

In a partitioned database environment, each buffer pool for a database has the same default definition on all database partitions (unless it was otherwise specified in the CREATE BUFFERPOOL statement, or the buffer pool's size was changed for a particular database partition with the ALTER BUFFERPOOL statement).

When you create a table space with a page size of 4 KB and do not assign it to a specific buffer pool, the table space is assigned to the default buffer pool. If you create a table space with a page size greater than 4 KB (8 KB, 16 KB, 32 KB) you should assign it to a buffer pool that uses a page size that is the same. If this buffer pool is currently not active, DB2 will attempt to assign the table space to an active buffer pool that uses an identical page size (if one is available). This assignment, if made, is temporary. When the database is activated again, and the originally specified buffer pool is active, then DB2 assigns the table space to that buffer pool.

You cannot use the ALTER TABLESPACE statement to add the table space to a buffer pool that uses a different page size.

When creating or altering buffer pools, the total memory that is required by all buffer pools must be available to the database manager so that all of the buffer pools can be allocated when the database is started. Should this memory not be available when a database is started, the Database Manager attempts to start the default buffer pool (IBMDEFAULTBP) and one of each buffer pool defined with a different page size, but only with a minimal size of 16 pages each. The size of this minimal buffer pool can be overridden with the registry variable *DB2_OVERRIDE_BPF*. See “Appendix A. DB2 Registry and Environment Variables” on page 469 for more information on this and other registry and environment variables. A warning message is returned with each failed attempt to start a buffer pool; the database continues in this operational state until its configuration is changed and the database can be fully restarted.

The reason for allowing the database manager to start with minimal-sized values is to allow you to connect to the database. You can then immediately reconfigure the buffer pool sizes; or, to perform other critical tasks. Do not consider operating the database for an extended time in such a state.

Note: Although the size and attributes associated with the default buffer pool can be changed, it cannot be dropped. Also, there is a minimum size for each buffer pool that is based on the platform being used.

There are advantages to having a large amount of memory allocated to buffer pools. For example, larger buffer pool sizes:
- Enable often-requested data pages to be kept in the buffer pool, allowing for quicker access. Fewer I/O operations can reduce I/O contention, thereby providing better response time and reducing the processor resource needed for I/O operations.
- Provide the opportunity to achieve higher transaction rates with the same response time.
- Prevent I/O contention for frequently used disk storage devices such as catalog tables and frequently referenced user tables and indexes. Sorts required by queries also benefit from reduced I/O contention on the disk storage devices containing the temporary table spaces.

Choosing One or Many Buffer Pools

If any of the following conditions apply to your system, you should use only a single buffer pool:
- The total buffer space is less than 10 000 4 KB pages.
- People with the application knowledge to do specialized tuning are not available.
- You are working on a test system.

If your system is not constrained by these conditions, then consider using more than one buffer pool for the following potential performance improvements:

- You can put temporary table spaces into a separate buffer pool to provide better performance for queries that require temporary storage, especially sort-intensive queries.
- If you have data that must be accessed repeatedly and quickly by many short update transaction applications, then you should consider moving the table space containing the data into a separate buffer pool. If this buffer pool is sized appropriately, its pages have a better chance of being found, contributing to a lower response time and a lower transaction cost.
- You can isolate data into separate buffer pools to favor certain applications, data, and indexes. For example, you might want to put tables and indexes that are updated frequently into a buffer pool that is separate from those tables and indexes that are frequently queried but infrequently updated. This change will reduce the impact of the frequent updates (on the first set of tables) on the frequent queries (on the second set of tables).
- You can use smaller buffer pools for the data accessed by applications that are seldom used, especially in the case where an application requires very random access into a very large table. In such a case, there is no need to keep the data in buffer pool memory for longer than a single query. It is better to keep a small buffer pool for this data, and free up the extra memory for other uses (for example, for other buffer pools).
- After separating different activities and data into separate buffer pools, good and relatively inexpensive performance diagnosis data can be produced from statistics and accounting traces.

Prefetching Data into the Buffer Pool

Prefetching index and data pages into the buffer pool can help improve performance by reducing the time spent waiting for I/O to complete. To *prefetch* pages means that one or more pages are retrieved from disk in anticipation of their use. There are two categories of prefetch:

- *Sequential prefetch* is a mechanism that reads consecutive pages into the buffer pool before the pages are required by the application. (See "Understanding Sequential Prefetching" on page 242.)
- *List prefetch*, or list sequential prefetch, is a way to access data pages efficiently, even when the data pages needed are not consecutive. (See "Understanding List Prefetching" on page 243.)

These two methods of reading data pages are in addition to a normal read. A normal read is used when just one or a few consecutive pages are retrieved. During a normal read, one page of data is transferred.

For further information on enabling prefetching, see also "Configuring I/O Servers for Prefetching and Parallel I/O" on page 244.

Understanding Sequential Prefetching

Reading several consecutive pages into the buffer pool using a single I/O operation can greatly reduce the overhead associated with running your application. In addition, performing multiple I/O operations in parallel to read in several ranges of pages at the same time can help reduce the time your application needs to wait for I/O operations to complete.

Prefetching is started when the Database Manager determines that sequential I/O is appropriate and that prefetching may help to improve performance. In cases such as table scans and table sorts, the Database Manager can easily determine that sequential prefetch will improve I/O performance. In these cases, the Database Manager automatically starts sequential prefetch. The following example could require a table scan and would be a good candidate for sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
```

The number of pages that the Database Manager will prefetch can be defined for each table space using the PREFETCHSIZE clause with either the CREATE TABLESPACE or ALTER TABLESPACE statements. The value specified is maintained in the PREFETCHSIZE column of the SYSCAT.TABLESPACES system catalog table.

It is a good practice to explicitly set the PREFETCHSIZE value as a multiple of the EXTENTSIZE value for your table space and the number of table space containers. (The extent size is the number of pages that the database manager writes to a container before using a different container; refer to “Designing and Choosing Table Spaces” in the *Administration Guide: Planning*.) For example, if the extent size is 16 pages and the table space has two containers, you could choose to set the prefetch quantity to 32 pages.

The Database Manager monitors buffer pool usage to ensure that prefetching of data does not remove pages from the buffer pool if those pages are needed by another unit of work. To avoid problems, the Database Manager may choose to limit the number of pages being prefetched to a quantity less than you specified for the table space.

The setting of the prefetch size can have significant performance implications, particularly for large table scans. You can use the database system monitor and other system monitor tools to help you tune PREFETCHSIZE for your table spaces. For example, you can gather information about whether:

- There are I/O waits for your query, using monitoring tools available for your operating system.

- Prefetch is occurring, by looking at the *pool_async_data_reads (buffer pool asynchronous data reads)* data element provided by the database system monitor. Refer to the *System Monitor Guide and Reference* for more information.

If there are I/O waits and the query is prefetching data, you can try increasing the value of PREFETCHSIZE. It is possible that the prefetcher is not the cause of the I/O wait, in which case increasing the PREFETCHSIZE value will not improve the performance of your query.

In all types of prefetch, multiple I/O operations may be performed in parallel when the prefetch size is a multiple of the extent size for the table space and the extents of the table space are in separate containers. For better performance the containers should be configured to use separate physical devices. For more information on parallel prefetching, see "Configuring I/O Servers for Prefetching and Parallel I/O" on page 244.

Understanding Sequential Detection

There are cases for which it is not immediately obvious whether sequential prefetch will improve performance. In these cases, the Database Manager can monitor I/O and if sequential page reading is occurring the Database Manager can activate prefetching. Prefetching in this case can be activated and deactivated by the Database Manager when it deems it appropriate. This type of sequential prefetch is known as *sequential detection* and applies to both index and data pages. You may use the *seqdetect* configuration parameter (see "Sequential Detection Flag (seqdetect)" on page 370) to control whether the Database Manager should perform sequential detection. If sequential detection is turned on, it could determine that the following SQL statement would benefit from sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
WHERE EMPNO BETWEEN 100 AND 3000
```

In this example, the optimizer may have chosen to scan the table using an index on the EMPNO column. If the table is highly clustered with respect to this index, then the data page reads will be almost sequential and prefetching may improve performance. In this case, data page prefetch will occur.

Index page prefetch may also occur in this example. If a large number of index pages have to be examined and the database manager detects that sequential page reading of the index pages is occurring, then index page prefetching will occur.

Understanding List Prefetching

List prefetch, or list sequential prefetch, is a way to access data pages efficiently, even when the data pages needed are not contiguous. List prefetch can be used in conjunction with either single or multiple index access.

Prefetching and Intra-Partition Parallelism

Prefetching is very important to the performance of intra-partition parallelism, which uses multiple subagents when scanning an index or a table. These parallel scans introduce larger data consumption rates, which require higher prefetch rates.

The cost of inadequate prefetching is higher for parallel scans than serial scans. If prefetching does not occur when executing a serial scan, the query runs more slowly because the agent always needs to wait for I/O. If prefetching does not occur when executing a parallel scan, all subagents may need to wait for one subagent that is waiting for I/O.

Because of its importance, prefetching is performed more aggressively with intra-partition parallelism. The sequential detection mechanism tolerates larger gaps between adjacent pages so that the pages can be considered sequential. The width of these gaps increases with the number of subagents involved in the scan.

Configuring I/O Servers for Prefetching and Parallel I/O

To enable prefetching, the Database Manager starts separate threads of control, known as *I/O servers*, to perform page reading. As a result, the query processing is divided into two parallel activities: data processing (CPU) and data page I/O. The I/O servers wait for prefetch requests from the CPU processing activity. These prefetch requests contain a description of the I/O needed to satisfy the anticipated data needs. The reason for prefetching determines when and how the Database Manager generates the prefetch requests. (See "Understanding Sequential Prefetching" on page 242 and "Understanding List Prefetching" on page 243 for more information.)

The following figure illustrates how I/O servers are used to prefetch data into a buffer pool.

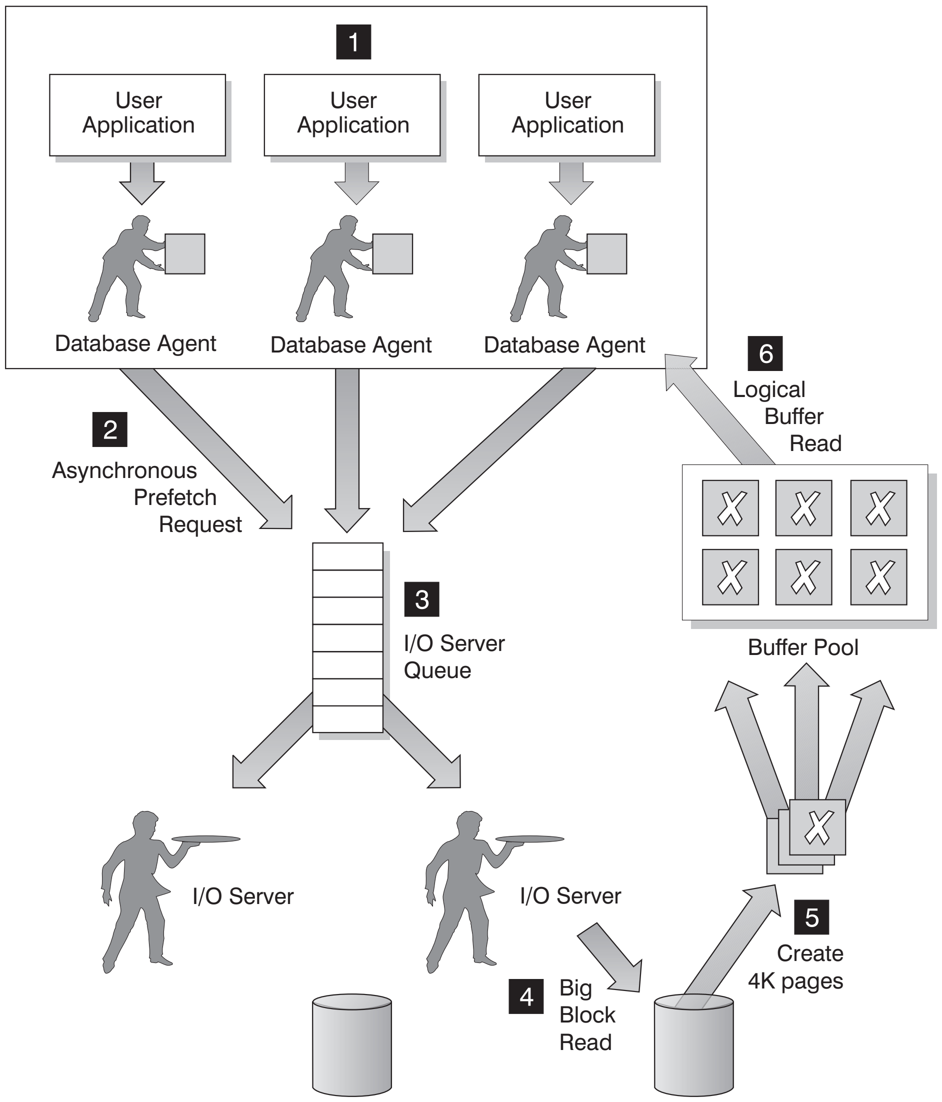


Figure 27. Prefetching Data using I/O Servers

The following steps are illustrated in Figure 27:

1 The user application passes the SQL request to the database agent.

2, **3**
The database agent determines that prefetching should be used to obtain the data required to satisfy the SQL request and writes a prefetch request to the I/O server queue.

4 , **5**
The first available I/O server will read the prefetch request from the queue and read the data from the table space into the buffer pool. Depending on the number of prefetch requests in the queue and the number of I/O servers configured by the *num_ioservers* configuration parameter, multiple I/O servers can be fetching data from the table space at the same time.

6 The database agent performs the necessary actions against the data pages in the buffer pool in order to return the result of the SQL request back to the user application.

Configuring enough I/O servers with the *num_ioservers* configuration parameter can greatly enhance the performance of queries for which prefetching of data can be used. Having some extra I/O servers configured will not hurt performance because extra I/O servers are not used and their memory pages will get paged out. Each I/O server process is numbered and the Database Manager will always use the lowest numbered process that is available and, as a result, some of the upper numbered processes may never be used.

To determine the number of I/O servers that you should configure, consider the following:

- The amount of concurrent activity against the database. That is, the number of database agents that could be writing prefetch requests to the I/O server queue at any given time.
- The highest degree to which the I/O servers can work in parallel. For more information, see "Enabling Parallel I/O".

To maximize the opportunity for parallel I/O, set *num_ioservers* to at least the number of physical disks in the database.

Enabling Parallel I/O

For situations in which multiple containers exist for a table space, the Database Manager can initiate *parallel I/O*. Parallel I/O refers to the ability of the Database Manager to use multiple I/O servers to process the I/O requirements of a single query. Each I/O server is assigned the I/O workload for a separate container, allowing several containers to be read in parallel. Performing I/O in parallel can result in significant improvements to I/O throughput.

While a separate I/O server will handle the workload for each container, the actual number of I/O servers that can perform I/O in parallel will be limited to the number of physical devices over which the requested data is spread. This also means you need as many I/O servers as the number of physical devices.

How parallel I/O is initiated and used is dependent on the reason for performing the I/O:

- **Sequential prefetch**

  For sequential prefetch, parallel I/O is initiated when the prefetch size is a multiple of the extent size for a table space. Each prefetch request is then broken into multiple, smaller, requests along the extent boundaries. These smaller requests are then assigned to different I/O servers.
- **List prefetch**

  For list prefetch, each list of pages is divided into smaller lists according to the container in which the data pages are stored. These smaller lists are then assigned to different I/O servers.
- **Database or table space backup and restore**

  For backing up or restoring data, the number of parallel I/O requests are equal to the backup buffer size divided by the extent size up to a maximum value equal to the number of containers.
- **Database or table space restore**

  For restoring data, the parallel I/O requests are initiated and split in a manner that is the same as that used for sequential prefetch. Instead of restoring the data into the buffer pool, the data is moved directly from the restore buffer to disk.
- **Load**

  When loading data you can specify the level of I/O parallelism with the LOAD command's DISK_PARALLELISM option. (If it is not specified, a default is used based on the cumulative number of table space containers for all table spaces associated with the table.)

For optimal performance of parallel I/O, ensure that:

- There are enough I/O servers. You should configure the number of I/O servers to be slightly higher than the number of containers used for all table spaces within the database.
- The extent size and prefetch size are sensible for the table space. Prefetch size should not be too large, to prevent over-use of the buffer pool. (An ideal size is a multiple of the extent size and the number of table space containers.) The extent size should be fairly small, with a good value being in the range of 8 to 32 pages.
- The containers are configured to reside on separate physical drives.
- All containers are the same size to ensure a consistent degree of parallelism.

  If one or more containers are smaller than the others, they will reduce the potential for optimized parallel prefetch. For example:
  - After a smaller container is filled up, additional data is stored in the remaining containers, causing the containers to become unbalanced. Unbalanced containers reduce the performance of parallel prefetching,

because the number of containers from which data can be prefetched may be less than the total number of containers.
    - If a smaller container is added at a later date and the data is rebalanced, the smaller container will contain less data than the other containers. Its small amount of data relative to the other containers will not optimize parallel prefetching.
    - If one container is larger and all of the other containers fill up, it will be the only container to store additional data. The database manager will not be able to use parallel prefetch to access this additional data.
- There is adequate I/O capacity when using intra-partition parallelism. Intra-partition parallelism can be used on SMP machines to reduce a query's elapsed time by running the query on multiple processors. Sufficient I/O capacity is required to keep each processor busy, usually requiring additional physical drives to provide the I/O capacity.

  Prefetching must occur at higher rates to use I/O capacity effectively. The prefetch size should be higher for prefetching to occur at higher rates. The prefetch size should be a multiple of the extent size and the number of table space containers. Ideally, containers should be configured to reside on separate physical drives.

  The number of physical drives required could depend on the speed and capacity of the drives and the I/O bus, and on the speed of the processors.

Allocating Multiple Pages at a Time

SMS table spaces are expanded on demand. This expansion is done a single page at a time by default. However, in certain work loads (for example, when doing a bulk insert) you can increase performance by using the *db2empfa* tool to tell DB2 to expand the table space in groups of pages or extents. The *db2empfa* tool is located in the *bin* subdirectory of the *sqllib* directory. Running it causes the *multipage_alloc* database configuration parameter to be set to YES. For more information on this tool, refer to the *Command Reference*.

Another way to make the best use of your available memory is discussed in "Extending Memory" on page 265.

Sorting

Sorting is often required for a query, and the proper configuration of the sort heap areas can be crucial to the query's performance. Sorting is required when:

- No index exists to satisfy a requested ordering (for example a SELECT statement that uses the ORDER BY clause)
- An index exists but sorting would be more efficient than using the index
- Creating an index (if the *indexsort* configuration parameter is set to yes).

Different Types of Sorting

Sorting involves two steps:

1. A sort phase
2. Return of the results of the sort phase.

How the sort is handled within these two steps results in different categories or types by which we can describe the sort. When considering the sort phase, the sort can be categorized as “overflowed” or “non-overflowed”. When considering the return of the results of the sort phase, the sort can be categorized as “piped” or “non-piped”.

Overflowed and Non-Overflowed
If the information being sorted cannot fit entirely into the sort heap (a block of memory that is allocated each time a sort is performed) it overflows into temporary database tables. Sorts that do not overflow always perform better than those that do.

Piped and Non-Piped
If sorted information can return directly without requiring a temporary table to store a final, sorted list of data, it is referred to as a “piped sort”. If the sorted information requires a temporary table to be returned, it is referred to as a “non-piped sort”. A piped sort always performs better than a non-piped sort.

Tuning the Parameters that Affect Sorting

The following situations affect the performance of sorting:

- The settings for the following configuration parameters:

  “Sort Heap Size (sortheap)” on page 342
  Specifies the amount of memory to be used for each sort

  “Sort Heap Threshold (sheapthres)” on page 342
  Controls the total amount of memory for sorting available across the entire instance for all sorts.
- Statements that involve a large amount of sorting
- Missing indexes that could help avoid unnecessary sorting
- Application logic that does not minimize sorting
- Parallel sorting, which improves the performance of sorts but can only occur if the statement uses intra-partition parallelism (see “Enabling Parallel I/O” on page 246).

Looking for Indicators of Sorting Performance Problems

To tell if you have an overall problem with sorting, look at the total CPU time spent sorting compared to the time spent on the whole application. The database system monitor can help (see “Using the Database System Monitor” on page 263). In particular, the Performance Monitor (which is made up of the “Snapshot Monitor” and “Event Monitor” and is available from the Control Center), shows *total sort time* by default, along with other times such as *I/O* and *lock wait*.

If total sort time is a large proportion of the other times then look at the following values, which are also shown by default:

Percentage of overflowed sorts
This variable (on the performance details view of the Snapshot Monitor) shows the percentage of sorts that overflowed. If the percentage of overflowed sorts is high, increase the *sortheap* and/or *sheapthres* configuration parameters if there were any post-threshold sorts. (To determine if there were any post threshold sorts, use the Snapshot Monitor.)

Post threshold sorts
If post threshold sorts are high, increase *sheapthres* and/or decrease *sortheap*.

In general, make the overall sort memory available across the instance (*sheapthres*) as large as possible without causing excessive paging. It is possible for a sort to be done entirely in sort memory. However, if this causes the operating system to perform excessive page swapping to accommodate that sort memory you can lose the advantage of a large sort heap. So, whenever you adjust the sorting configuration parameters, use an operating system monitor to track any changes in system paging.

Note: With the improvement in the DB2 partial key binary sorting technique to include non-integer data type keys, some additional memory is required when sorting long keys. If you believe long keys are being used, increase the *sortheap* configuration parameter.

Also note that in a piped sort, the sort heap does not get freed until the application closes the cursor associated with that sort. So a piped sort can use up memory until the cursor is closed.

Techniques for Managing Sorting Performance

You can use the database system monitor and benchmarking techniques to help set the *sortheap* and *sheapthres* configuration parameters. Do the following for each database manager and its databases:

- Set up and run a representative workload.
- For each applicable database, collect average values for the following performance variables over the benchmark workload period:
  - Total sort heap in use
  - Active sorts

These performance variables are shown on the performance details view of the Snapshot Monitor.

- Set *sortheap* to the average *total sort heap in use* for each database.
- Set the *sheapthres* by doing the following:
  1. Determine which database in the instance has the largest *sortheap* value.
  2. Determine the average size of the sort heap for this database.

     If this is too difficult to determine, use 80% of the maximum sort heap
  3. Set *sheapthres* to the average number of active sorts times the average size of the sort heap computed above.

     This is a recommended initial setting. You can then use benchmark techniques to refine this value.

You can also identify particular applications and statements where sorting is a significant performance problem:

- Set up event monitors at the application and statement level to help you identify applications with the longest total sort time.
- Within each of these applications, find the statements with the longest *total sort time*.
- Tune these statements using a tool such as Visual Explain.
- Ensure that appropriate indexes exist. You can use Visual Explain to identify all the sort operations for a given statement. Then investigate whether or not an appropriate index exists for each table accessed by the statement.

Note: You can search through the explain tables to identify which queries have sort operations. (See "Appendix C. SQL Explain Tools" on page 531.)

Reorganizing Catalogs and User Tables

The performance of SQL statements that use indexes can be impaired after many updates, deletes, or inserts have been made. Generally, newly inserted rows cannot be placed in a physical sequence that is the same as the logical sequence defined by the index (unless you use clustered indexes). This means that the Database Manager must perform additional read operations to access the data, because logically sequential data may be on different physical data pages that are not sequential.

In general, reorganizing a table takes more time than running statistics. Performance may be improved sufficiently by obtaining the current statistics for your data and rebinding your applications, so try this first. If this does not improve performance, the data in the tables and indexes may not be arranged efficiently, so reorganization may help. The information in this section applies not only to reorganizing your own tables, but also to the system catalog tables which may also require reorganization.

For typed tables, the specified table name must be the name of the hierarchy's root table.

The REORGCHK command returns information about the physical characteristics of a table, and whether or not it would be beneficial to reorganize that table. This command can be used through the command line processor. Refer to the *Command Reference* for more information, including how to interpret the command output.

Note: The REORGCHK command does not show any data for extended indexes, nor for declared temporary tables.

The REORG utility optionally rearranges data into a physical sequence according to a specified index. REORG has an option to specify the order of rows in a table with an index, thereby clustering the table data according to the index and improving the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility. As a result, SQL statements requiring rows in the indexed order can be processed more efficiently. REORG also stores the tables more compactly by removing unused, empty space (though if you specified PCTFREE when you used ALTER TABLE, that space remains unused).

Do not use the REORG or REORGCHK commands with nicknames.

The REORG utility requires that all other applications that would normally be working against the affected table data and indexes be offline. You may have a work environment where you wish to limit the amount of time your applications cannot work against the data. In this environment, you might consider using the online index reorganization utility.

You may wish to consider the following factors to determine when to reorganize your table data:
- The volume of insert, update, and delete activity
- Any significant change to the performance of queries which use an index with a high cluster ratio
- Running statistics (RUNSTATS) does not improve the performance of queries
- The REORGCHK command indicates a need to reorganize your table
- The cost of reorganizing your table, including the CPU time, the elapsed time, and the reduced concurrency resulting from the REORG utility locking the table until the reorganization is complete.

To execute the REORG utility, you must have SYSADM, SYSMAINT, SYSCTRL or DBADM authority, or CONTROL privilege on the table.

The REORG utility uses temporary tables that can be significantly larger than the original table, if columns were added to a table, or a table has LOB columns. If these temporary tables are larger, the resulting permanent table, created by the REORG utility, will also be larger.

For typed tables, the specified table name must be the name of the hierarchy's root table.

Note: You cannot use the REORG utility to reorganize declared temporary tables.

The REORG utility allows you to specify a temporary table space, which is used to create the temporary REORG table. If a temporary table space is not specified, the REORG utility will create the temporary REORG tables in the table space that contains the table being reorganized. The following guidelines can assist you in determining whether to use a temporary table space:

- If you specify a temporary table space, it is generally recommended that you specify an SMS temporary table space. A DMS temporary table space is not recommended since you can only have one REORG in progress using this type of table space.
- It is generally recommended that you specify a temporary SMS table space. Using a temporary DMS table space is generally not recommended since you can only have one REORG in progress using this type of table space.

The REORG utility implicitly closes all open cursors.

Remember that you may be reorganizing a table within a table space that is using greater than 4 KB pages (8 KB, 16 KB, or 32 KB) pages. During the reorganization, the temporary table space used during the reorganization must have the same size pages as the base table space.

If the REORG utility does not complete successfully, do **not** delete any temporary files, tables or table spaces. These files and tables are used by the Database Manager to roll back the changes made by the REORG utility, or to complete the reorganization, depending on how far the reorganization had progressed before the failure.

In a partitioned database, the REORG utility reorganizes data on each partition. If the utility fails on any partition, only the failing partition is rolled back. If you specify a directory path to store temporary tables, this path is extended by the Database Manager at each database partition. Therefore, if you specify a path that is shared by other database partitions, the temporary files are stored in different subdirectories (identified by node name) under this path.

Online Index Reorganization

An online reorganization is possible by providing a user-definable threshold for the maximum amount of free space on an index leaf page. When there is a deletion of an index key from a leaf page and the threshold is crossed, the neighboring index leaf pages are checked to determine if two leaf pages can be merged. If there is sufficient space on a page for a merge of two neighboring pages to take place, the merge occurs without having to take the database offline.

This online reorganization of the index is only possible with indexes created in this release and those following this release. Existing indexes requiring the ability to reorganize online in this fashion will have to be dropped and then re-created in order for the necessary internal changes to the index leaf pages. To turn on online index reorganization for a particular index, specify a `MINPCTUSED` value when the index is created. The `MINPCTUSED` value should be set to less than one hundred (100). This value becomes the reorganization threshold which is the percentage of used space on an index page that is the minimum acceptable value before attempting to merge the index leaf page with that of it's neighbor. The recommended value for `MINPCTUSED` is one that is less than 50 percent since the goal is to merge two neighboring index leaf pages. A value of zero for `MINPCTUSED`, which is also the default, disables online reorganization.

Index leaf pages that are freed for use following an online index reorganization are available for re-use. However, the freed pages are available only to other indexes in the same table. A full reorganization of the table will free up pages for other object when working with a DMS storage model; or will free up disk space when working with a SMS storage model.

Index non-leaf pages are not freed for use following an online index reorganization. However, a full reorganization of the table will make the index as small as possible. The leaf and non-leaf pages are reduced in number as well as the levels of the index.

Avoiding the Need to Reorganize Tables

To reduce the need for reorganizing a table, do the following after you have created the table:
- Alter table to add PCTFREE
- Create clustering index with PCTFREE on index
- Sort the data
- Load the data.

Now you have a table with a clustering index. The clustering index, in conjunction with PCTFREE on table, will preserve the original sorted order. With sufficient space on pages, new data can be inserted on the correct pages thereby maintaining the clustering characteristics of the clustering index. If, as more data is inserted, and the pages of the table become full, records are appended to the end of the table, and the table gradually becomes unclustered.

It is recommended that you perform a REORG or a sort and LOAD after creating a clustering index. A clustering index attempts to maintain a particular order of data improving the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility.

The amount of free space to be left on each page during a REORG is determined by the PCTFREE value of the table. If this value has not been set, REORG will fill up the pages as the data is being reorganized.

Performance Considerations for DMS Devices

If you are using Database Managed Storage (DMS) device containers for your table spaces, you need to understand the following so you can effectively administer your environment:

- **File system caching**

  File system caching is performed as follows:

  - For DMS file containers (and all SMS containers), the operating system may cache pages in the file system cache
  - For DMS device container table spaces, the operating system does not cache pages in the file system cache.

    Note: When working on Windows NT, the registry variable DB2NTNOCACHE specifies whether or not DB2 will open database files with a NOCACHE option. If DB2NTNOCACHE=ON, file system caching is eliminated. If DB2NTNOCACHE=OFF, the operating system caches DB2 files. This applies to all data except for files that contain LONG FIELDS or LOBS. Eliminating system caching allows more memory to be available to the database so that the buffer pool or sortheap can be increased.

- **Buffering of data**

  Table data read from disk is normally available in the database's buffer pool (see "Managing the Database Buffer Pool" on page 235). In some cases, a data page can be freed from the buffer pool before the application has actually used that page. (This can happen if the buffer pool space is required for other data pages.) For table spaces using system managed storage (SMS) or database managed storage (DMS) file containers, see the description of file system caching above. This can eliminate I/O that would otherwise have been required.

Table spaces using database managed storage (DMS) device containers do **not** use the file system or its cache. As a result, you may wish to increase the size of the database buffer pool and reduce the size of the file system cache to offset the fact that double buffering is not being done with DMS table spaces that use device containers.

If you notice, through the use of system-level monitoring tools, that I/O is higher for a DMS table space using device containers compared to the equivalent SMS table space, this difference could be due to the double buffering discussed above.

- **Using LOB or LONG data**

  When an application retrieves either LOB or LONG data, the database manager does not use its buffers to cache the data. Every time an application needs one of these pages, the database manager must retrieve it from disk.

  However, if LOB or LONG data is stored in SMS or DMS file containers, file system caching may provide buffering and, as a result, better performance.

  Because system catalogs contain some LOB columns, it is recommended that you keep them in SMS (or alternatively in DMS-file) table spaces.

Managing Initialization Overhead

The ACTIVATE DATABASE command starts up selected databases. Using this command in a partitioned database results in an attempt to activate the selected partitioned database on all database partitions. By using this command, no application time is spent on database initialization or startup.

Databases that you have initialized using the ACTIVATE DATABASE command must be shut down with the DEACTIVATE DATABASE command; the last application disconnecting from the database will not shut it down. For more information on the ACTIVATE and DEACTIVATE commands, refer to the *Command Reference* manual.

If a database has not been started, and a CONNECT TO (or an implicit connect) is encountered in an application, then the application must wait while the Database Manager starts up the required database before it can do any work with that database. This is a startup cost that is borne by the first application to access a particular database. In a partitioned database, this startup cost is incurred on each database partition. Once the database is started, all other applications can connect to and use the database without a time cost associated with the database startup.

Database Agents

DB2 servers must facilitate communication between the database manager and client and local applications. UNIX-based environments use an architecture based on *processes*. For example, the DB2 communications listeners are created as processes. Intel operating systems such as OS/2 and Windows NT use an architecture based on *threads* to maximize performance. For example, the DB2 communications listeners are created as threads within the DB2 server's system controller process. For each database being accessed, various processes/threads are started to deal with the various database tasks (for example, prefetching, communication, and logging).

One of the most crucial processes/threads are those of database agents, which facilitate the operations of applications with databases.

A *logical agent* represents a connected application to the database manager. The logical agent has all the information and control blocks required by an application. The maximum number of logical agents is contolled by the *max_logicagents* database manager configuration parameter. Since each application will have one logical agent, this parameter controls the maximum number of applications that can be connected to the instance.

A *worker agent* carries out application requests but has no permanent attachment to any particular application. The worker agent has all the information and control blocks required to complete actions within the database manager that were requested by the application.

There are four types of worker agents: *active coordinator agents*, *subagents*, *inactive agents*, and *idle agents*.

The idle agent is the simplest form of worker agent: It is not tied to a logical agent, it does not have an outbound connection, and it does not have a local database connection or an instance attachment.

The inactive agent is a worker agent which is not in an active transaction, is not tied to a logical agent, does not have an outbound connection, and does not have a local database connection or an instance attachment. An inactive agent is free to tie to another logical agent to begin serving the application represented by that logical agent.

Each process/thread of a client application has a single *active coordinator agent* that operates on a database. Once the coordinator agent is created, it performs all database requests on behalf of its application, and communicates to other agents using inter-process communications (IPC) or remote communication protocols. Each agent operates with its own private memory and shares Database Manager and database global resources such as the buffer pool with other agents. When a transaction completes, the active coordinator agent may detach from the logical agent and thus become an inactive agent.

In partitioned database environments and environments with intra-partition parallelism enabled, the coordinator agent distributes database requests to *subagents*, and these agents perform the requests for the application. Once the coordinator agent is created, it handles all database requests on behalf of its application by coordinating the subagents that perform requests on the database.

When a client disconnects from a database or detaches from an instance the coordinating agent will be:

- An active agent. If other logical agents are waiting, the worker agent will become an active coordinator agent.
- Freed and marked as idle, if no other logical agents are waiting and the maximum number of pool agents has not been reached.
- Terminated and its storage freed, if no other logical agents are waiting and the maximum number of pool agents has been reached.

Those agents not performing work on behalf of any applications and who are waiting to be assigned, are considered to be idle agents and reside in an *agent pool*. These agents are available for requests from coordinator agents operating on behalf of client programs, or for subagents operating on behalf of existing coordinator agents. The number of available agents is dependent on the database manager configuration parameters *maxagents* and *num_poolagents*.

Agents from the agent pool (*num_poolagents*) are re-used as coordinator agents:

- For remote TCP/IP-based applications; or
- For local applications on UNIX-based operating systems; or
- For both local and remote applications on Windows NT and OS/2 operating systems.

Otherwise, remote applications always create a new agent.

If no idle agents exist when an agent is required, a new agent must be dynamically created. Creating a new agent involves a certain amount of overhead and as a result, improved CONNECT and ATTACH performance can be noticed if there is an idle agent that can be activated for a client.

When a subagent is working on behalf of an application, it is considered to be *associated* with that application. After completing the assigned work, it may be placed in the agent pool, but it remains associated with the original application. When the application requests additional work, the database manager first checks the idle pool for associated agents when finding an agent to work for the application.

The ability to separately control the number of connected applications (using the number of logical agents defined by *max_logicagents*) and the number of application requests that can be processed (using the number of active coordinator agents defined by *max_coordagents*) allows for flexibility in the workloads processed at the database. A one-to-one relationship between the number of connected applications and the number of application requests that can be processed is the typical way applications will work with the database. However, it may be that your work environment is such that you require a many-to-one relationship between the number of connected applications and the number of application requests that can be processed.

Since the database global resource overhead is associated with the active coordinator agents, the greater the number of these agents means there is a greater chance that the upper limits of available database global resources will be reached. You may want to allow more connected applications than active coordinator agents so that the upper limits of available database global resources are not reached. By setting the value of *max_logicagents* greater than the value for *max_coordagents*, you are concentrating your database work.

Refer to *DB2 Connect User's Guide* for more information and examples of how to use DB2 Connect as an XA transaction support concentrator.

When working in an environment requiring the use of DB2 Connect to connect to remote systems there is an *outbound connect pool*. This connection pool reduces the connect time (following the first connection) to a host. When a disconnection from a host is requested, DB2 Connect drops the inbound connection but keeps the outbound connection to the host in a pool. When a new request is made to connect to the host, DB2 Connect reuses an existing outbound connection (if available) from the pool.

Note: When using connection pooling, DB2 Connect is restricted to inbound TCP/IP and to outbound TCP/IP and SNA connections. When working with SNA, the security type must be `NONE` for the connection to be placed in the pool.

With connection pooling, the active agent does not close its outbound connection following disconnection, but goes into the agent pool with an active connection to the remote host. This type of agent is called *inactive DRDA agent*. The pool of inactive DRDA agents is a synonym for the outbound connection pool.

Consider the following examples based on four different usage and workload requirements:

1. In the first example, an average of 40 concurrent users connect to remote host databases through DB2 Connect. At times the number of concurrent connections peaks at about 50, but never exceeds 55. The transactions are of short duration, and user connect and disconnect frequently.

   With these conditions, the system administrator should configure MAX_COORDAGENTS to 55 since he knows that the maximum number of users what will ever try to connect through DB2 Connect at the same time is 55. NUM_POOLAGENTS, the size of the agent pool, should be set to 40 since, at any one time, that is the average number of users connected or trying to connect. This pool size guarantees enough existing remote database connections to satisfy all inbound clients without having to establish any new ones except when the workload peaks.
2. In this second example the workload is much higher with about 1 000 inbound clients. User connections are also of short duration. The system administrator does not want to allow any more concurrent connections than that. Therefore, the system administrator sets both MAX_COORDAGENTS and NUM_POOLAGENTS to 1 000. This means that the maximum number of inbound clients that may be concurrently connected to the remote database(s) is 1 000. When all clients disconnect, the pool will contain exactly 1 000 connected agents all waiting to service new inbound clients.
3. The third example involves a single application connecting through DB2 Connect to just one remote database. The application remains connected for long periods of time. In this scenario, the best agent and connection pool configuration is to set MAX_COORDAGENTS to 1 since we know that at most only one client will connect. NUM_POOLAGENTS may be set to zero in this case since there is no frequent connection and disconnection from the remote host. Setting NUM_POOLAGENTS to zero effectively disables connection pooling since no agents with active connections to the remote database are kept in the pool. For every new inbound client that connects, a new agent is created and a new remote connection established to service it.
4. The fourth example is a variation based on all three previous workload scenarios. In this example, the system administrator wants to restrict concurrent access to remote databases to just 100. Therefore, MAX_COORDAGENTS is set to 100 and, in order to maximize connect performance, NUM_POOLAGENTS is set to 100. However, later, there may also be a need to connect locally to monitor the workload on the system where DB2 Connect is installed. The expectation is that no more than 5 concurrent monitor snapshots would occur at any one time so MAX_COORDAGENTS is set to 105. This new configuration value allows the maximum number of concurrent applications to grow beyond the earlier upper limit of 100 to accommodate the occasional monitor snapshot and/or instance attachment.

For partitioned database environments and environments with intra-partition parallelism enabled, each partition (that is, each database server or node) has its own pool of agents from which subagents are drawn. Because of this pool, subagents do not have to be created and destroyed each time one is needed or is finished its work. The subagents can remain as associated agents in the pool and be used by the database manager for new requests from the application they are associated with.

The following database manager configuration parameters affect the number of database agents:

- “Maximum Number of Agents (maxagents)” on page 379. Once the number of worker agents reaches this value, all subsequent requests that require a new agent are denied until the number of agents falls below the value. This value applies to the total number of agents, including coordinating agents, subagents, inactive agents, and idle agents, that are working on all applications.
- “Agent Pool Size (num_poolagents)” on page 383. The number of inactive agents, idle agents, and associated subagents in the agent pool cannot exceed this value.
- “Initial Number of Agents in Pool (num_initagents)” on page 384. When the database manager is started, a pool of worker agents is created based on this value. This speeds up performance for initial queries. The worker agents all begin as idle agents.
- “Maximum Number of Logical Agents (max_logicagents)” on page 382. The maximum number of logical agents. Since each application will have one logical agent, this parameter controls the maximum number of applications that can be connected to the instance.
- “Maximum Number of Coordinating Agents (max_coordagents)” on page 381. For partitioned database environments and environments with intra-partition parallelism enabled, this value limits the number of coordinating agents.
- “Maximum Number of Concurrent Agents (maxcagents)” on page 380. This value controls the number of *tokens* permitted by the Database Manager. For each database transaction (unit of work) that occurs when a client is connected to a database, a coordinating agent must obtain permission to process the transaction (known as a processing token) from the Database Manager. Only agents with a processing token are permitted by the Database Manager to execute a unit of work against a database. If a token is not available, the agent will wait until one is available, at which time the requested unit of work will be processed.

  This parameter can be useful in an environment in which peak usage requirements exceed system resources (memory, CPU, and disk). In such an environment, the peak load may cause excessive performance degradation because of, for example, paging. You can use this parameter to control the load and avoid the performance degradation.

For partitioned database environments and environments with intra-partition parallelism enabled, the impact to performance and memory costs within the system is strongly related to how your agent pool is tuned:

- The database manager configuration parameter for agent pool size (*num_poolagents*) affects the total number of subagents that can be kept associated with applications on a partition (that is, node). If the pool size is too small (and the pool is full), a subagent will disassociate itself from the application it worked on and terminate. This situation leads to poor performance, because subagents must be constantly created and reassociated to applications.

  In addition, if the value of *num_poolagents* is too small, one application may fill the pool with associated subagents. Thus, when another application requires a new subagent and has no subagents in its associated agent pool, it will "steal" subagents from the agent pools of other applications. This situation is rather costly, and causes poor performance.
- The above situations must be weighed against the resource costs of allowing too many agents to be active at any given time.

  For example, if the value of *num_poolagents* is too large, associated subagents may sit unused in the pool for long periods of time. These subagents use database manager resources that will not be available for other tasks.

In addition to the database agents, there are other asynchronous activities performed by the Database Manager which run as their own process (or thread), including:

- Database I/O servers (or I/O prefetchers) (see "Prefetching Data into the Buffer Pool" on page 241)
- Database asynchronous page cleaners (see "Managing the Database Buffer Pool" on page 235)
- Database loggers
- Database deadlock detectors
- Event monitors
- Communication and IPC listeners
- Table space container rebalancers.

For more information on identifying the various DB2 processes, refer to the *Troubleshooting Guide*.

Using the Database System Monitor

The DB2 database manager maintains data about its operation, its performance, and the applications using it. This data is maintained as the database manager runs, and can provide important performance and troubleshooting information. For example, you can find out:

- The number of applications connected to a database, their status, and which SQL statements each application is executing, if any.
- Information that shows how well the database manager and database are configured, and helps you to tune them.
- When deadlocks occurred for a specified database, which applications were involved, and which locks were in contention.
- The list of locks held by an application or a database. If the application cannot proceed because it is waiting for a lock, there is additional information on the lock, including which application is holding it.

Because collecting some of this data introduces overhead on the operation of DB2, **monitor switches** are available to control which information is collected. To set monitor switches explicitly, use the UPDATE MONITOR SWITCHES command or the sqlmon() API. (You must have SYSADM, SYSCTRL, or SYSMAINT authority.)

There are two ways to access the data maintained by the database manager:

- **Taking a snapshot**

  Use the GET SNAPSHOT command from the command line; the Control Center on the OS/2, Windows 95, or Windows NT operating systems for a graphical interface; or write your own application, using the sqlmonss() API call.

  The Control Center, available from the DB2 folder or with the `db2cc` command, provides a performance monitor tool that samples monitor data at regular intervals by taking snapshots. This graphical interface provides either graphs or textual views of the snapshot data, in both detail and summary form. You can also define performance variables using data elements returned by the database monitor.

  The Control Center's Snapshot Monitor tool also allows you to define exception conditions by specifying threshold values on performance variables. When a threshold value is reached, you can predefine any of the following actions to occur: notification through a window or audible alarm, and/or execution of a script or program.

  If you are taking a snapshot from the Control Center, you cannot perform an action that either alters, changes, or deletes a database object (such as an instance or database) while you are performing snapshot monitoring on either that object, or on any it its child objects. (In addition, if you are monitoring a partitioned database system, you cannot refresh the view of partitioned database objects.) For example, you cannot monitor database A if you want to remove its instance. If, however, you are monitoring the instance only, you can alter database A.

To stop all monitoring for an instance (including any of its child objects), select **Stop all monitoring** from the pop-up menu for the instance. You should always stop monitoring from the instance, as this ensures that all locks that are held by the performance monitor are released.

- **Using an event monitor**

  An event monitor captures system monitor information after particular events have occurred, such as the end of a transaction, the end of a statement, or the detection of a deadlock. This information can be written to files or to a named pipe.

  To use an event monitor:

  1. Create its definition with the Control Center or the SQL statement CREATE EVENT MONITOR. This statement stores the definition in database system catalogs.
  2. Activate the event monitor through the Control Center, or with the SQL statement:

     ```
     SET EVENT MONITOR evname STATE 1
     ```

     If writing to a named pipe, start the application reading from the named pipe before activating the event monitor. You can either write your own application to do this, or use **db2evmon**. Once the event monitor is active and starts writing events to the pipe, **db2evmon** will read them as they are being generated and write them to standard output.
  3. Read the trace. If using a file event monitor, you can view the binary trace that it creates in either of the following ways:
     – Use the **db2evmon** tool to format the trace to standard output.
     – Click on the **Event Analyzer** icon in the Control Center (on the Windows 95, Windows NT, or OS/2 systems) to use a graphical interface to view the trace, search for keywords, and filter out unwanted data.

     Note: If the database system that you are monitoring is not running on the same machine as the Control Center, you must copy the event monitor file to the same machine as the Control Center before you can view the trace. An alternative method is to place the file in a shared file system accessible to both machines.

For information on the system database monitor and the event monitor, refer to the *System Monitor Guide and Reference*.

Extending Memory

Your machine may have more real addressable memory than the maximum amount of virtual addressable memory (for example, virtual addressable memory is usually between 2 GB and 4 GB on most platforms). You can configure any additional real addressable memory beyond virtual addressable memory as an *extended storage cache*. Such an extended storage cache can be used by any of the defined buffer pools and should improve the performance of the database manager. The extended storage cache is defined in terms of memory segments.

You should be aware when deciding to use some of the real addressable memory as an extended storage cache that this memory can then no longer be used for other purposes on the machine such as a jfs-cache or as process private address space. Assigning additional real addressable memory to the extended storage cache could lead to higher system paging.

DB2 makes use of addressable memory in your machine with buffer pools (see "Managing the Database Buffer Pool" on page 235). The extended storage cache is used by the buffer pools as a secondary level of caching (with the buffer pools performing the first level of caching). Ideally buffer pools can hold the data that is most frequently accessed, while the extended storage cache can hold data that is accessed, but less frequently.

The following database configuration parameters influence the amount and the size of the memory available for extended storage:

- *num_estore_segs* defines the number of extended storage memory segments. The default for this configuration parameter is zero, which specifies that no extended storage cache exists. (See "Number of Extended Storage Memory Segments (num_estore_segs)" on page 373.)
- *estore_seg_sz* defines the size of each extended memory segment. This size is limited by the platform on which the extended storage cache is being used. (See "Extended Storage Memory Segment Size (estore_seg_sz)" on page 372.)

Because an extended storage cache is an extension to a buffer pool, it must always be associated with one or more specific buffer pools. Therefore, you must declare which buffer pools can take advantage of a cache once it is created. The CREATE and ALTER BUFFERPOOL statements have the attributes NOT EXTENDED STORAGE and EXTENDED STORAGE that control cache usage. By default neither IBMDEFAULTBP nor any newly created buffer pool will use extended storage.

Note: If you are using buffer pools defined with different page sizes then the extended storage support for buffer pools is deactivated.

The database manager cannot directly manipulate data that resides in the extended storage cache. However, it can transfer data from the extended storage cache to the buffer pool much faster than from disk storage.

When a row of data is needed from a page in an extended storage cache, the entire page is read into the corresponding buffer pool.

A buffer pool and its associated extended storage cache, if defined, are created when a database is activated or first connected to.

Chapter 9. Using the Governor

You use the governor to monitor and change the behavior of applications that run against a database.

The governor consists of two parts:
- A front-end utility
- A daemon

When you start the governor, you issue a start command from the governor front-end utility, which then starts the governor daemon. By default, a daemon is started on every partition in a partitioned database, but you can also use the front-end utility to start a single daemon at a specific partition to monitor the activity against the database partition found there. Or, a daemon can monitor the activity on a single-partition database. See "Starting and Stopping the Governor" for details.

Each governor daemon collects statistics about the applications running against a database. It then checks these statistics against the rules that you specified in a governor configuration file that applies to that specific database. (See "Creating the Governor Configuration File" on page 270 for details.) The governor then acts according to these rules. For example, a rule may indicate that an application is using too much resource. In this situation, the governor may change the application's priority or force it off the database, according to the instructions you specified in the governor configuration file.

If the action associated with a rule is to change the application's priority, the governor changes the priority of agents on the database partition on which the governor detected the resource violation. If the action associated with a rule is to force an application, the application is forced even if the governor that detected the resource violation is running on the application's coordinator node or in a partitioned database environment.

The governor also logs any actions that it takes. You can query the log files to review the actions that the governor has taken. For details, see "Governor Log Files" on page 278 and "Querying Governor Log Files" on page 279.

Starting and Stopping the Governor

You use the `db2gov` governor front-end utility to start or stop the governor (on either all database partitions or on a single database partition). You require SYSADM or SYSCTRL authority to use the utility.

The syntax for db2gov is as follows:

```
►►─db2gov──┬─start─database──┬──────────────────────┬──config-file─log-file──┬──────►◄
           │                 └─nodenum─node-num─────┘                         │
           └─stop─database──┬──────────────────────┬──────────────────────────┘
                            └─nodenum─node-num─────┘
```

Figure 28. Syntax for db2gov

The parameters are as follows:

start **database**
: Starts the governor daemon to monitor the specified database. For *database*, you can specify either the database name or the database alias.

  The database name you specify must be the same name as that specified in the governor configuration file. The governor checks these two names to ensure that you are using the correct configuration file. If the front-end utility is started with one alias name and the governor configuration file contains a different alias, an error is reported because the governor cannot determine whether the names are aliases for the same database.

  If you are in a partitioned database environment, when you start the governor on all partitions, the front-end utility first checks that the configuration file does not contain errors. It then reads the node configuration file and sends a command to each database partition to start the governor front-end utility on each database partition with the start option (which, in turn, starts the daemon at each database partition).

  Note: Because the governor monitors at the database level, one daemon runs for each database that is being monitored. (In a partitioned database environment, one daemon runs for each database partition.) If the governor is running for more than one database, there will be more than one daemon running at that database server.

nodenum **node-num**
: Specifies the database partition on which to start the governor daemon. The number is the same as that specified in the node configuration file.

  When you start the governor on a single database partition, the front-end utility creates a daemon to validate the governor configuration file. The governor daemon ensures that another daemon is not already running on that partition.

config-file
Specifies the configuration file to use when monitoring the database.

The default location for the configuration file is the `sqllib` directory. If the specified file is not there, the front-end assumes that the specified name is the full name of the file.

log-file
Specifies the base name of the file to which the governor writes log records. The log file is stored in the `log` subdirectory of the `sqllib` directory. (On Windows NT, the `log` subdirectory is under the instance directory.) The number of the database partition on which the governor is running is automatically appended to the log file name (for example, `mylog.0, mylog.1, mylog.2`).

stop **database**
Stops the governor daemon that is monitoring the specified database.

If you are in a partitioned database environment, the front-end utility stops the governor on all database partitions by reading the node configuration file, and then sending a command to each database partition to call the governor front-end utility with the stop parameter. This stops the daemon at each database partition.

nodenum **node-num**
Specifies the database partition on which to stop the governor daemon. The number is the same as that specified in the node configuration file.

When the front-end utility stops the governor daemon on a single database partition, it communicates with the daemon on that database partition by creating, moving, or deleting files in the `tmp` subdirectory of the `sqllib` directory. You should not attempt to delete or modify these files.

The Governor Daemon

When the governor daemon is started (either by the `db2gov` front-end utility or by waking up), it runs in a loop. The first task it does is to check whether its governor configuration file has changed or has not yet been read. If either condition is true, the daemon reads the rules in the file. This allows you to change the behavior of the governor daemon while it is running.

After this, the governor daemon issues a snapshot request to obtain statistics for each application and agent working on the database.

Note: On some platforms, the CPU statistics are not available from the DB2 Monitor. Where this is the case, the account rule and the CPU limit will not be available.

The governor then checks the statistics for each application against the rules in the governor configuration file. If a rule applies to an application, the governor can: force the application; change the application's priority, which indirectly changes all the agent priorities of both agents and subagents that are working for it on that node; or, change the schedule for the application which, indirectly changes the agent priorities working on the application, depending on the action specified by the rule. The governor writes a record of any action it takes to a log file.

Note: The governor cannot be used as an alternate means to adjust agent priorities if the *agentpri* database manager configuration parameter is anything other than the system default. (This note does not apply to OS/2 or Windows NT platforms.)

When the governor finishes checking all of the applications, it sleeps for the interval specified in the configuration file. Once this time has elapsed, the governor wakes up and begins the execution loop again.

When the governor encounters an error or stop signal, it does cleanup processing before ending. The cleanup processing resets all application agent priorities (using a list of applications whose priorities have been set). It then resets the priorities of any agents that are no longer working on an application. This ensures that agents do not remain running with nondefault priorities after the governor ends. If an error occurs, a message is written to the `db2diag.log` file to indicate that the governor ended abnormally.

Note: The governor daemon is not a database application, and, therefore, does not maintain a connection to the database. (It does have an instance attachment, however.) The governor daemon can detect when the database manager ends because it can issue snapshot requests.

Creating the Governor Configuration File

When you start the governor, you specify the name of the configuration file that contains the rules to be used to govern applications running against the database. The governor acts based on these rules.

If your requirements for governing the database change, you can edit the configuration file without stopping the governor. Each governor daemon will detect that the file has changed, and reread it.

You must create the configuration file in a directory that is mounted across all the database nodes, because the governor daemon on each node must be able to read the same configuration file.

The configuration file consists of rules and comments. Most entries can be specified in uppercase, lowercase, or mixed case characters. The exception is `applname` which is case sensitive.

You delimit comments within the { } braces. The rules include:
- The database to which the rules apply.
- The length of time the governor sleeps before waking up to check the applications.
- The rules that specify how to govern the applications. These rules are made of smaller components called rule clauses.

Each rule in the file must be followed by a semicolon (;).

The following rules specify the database being monitored, and the interval at which the daemon wakes up after working through its loop of activities (which are described in “The Governor Daemon” on page 269). Each of these rules are only specified once in the file.

dbname
The name or alias of the database to be monitored.

account *nnn*
Account records are written containing CPU usage statistics for each connection at the specified number of minutes.

Note: This option is not available in Windows NT or OS/2 environments.

If a short connect session occurs entirely within the account interval, no log record is written. When log records are written, they contain CPU statistics that reflect CPU usage since the previous log record for the connection. If the governor is stopped then restarted, CPU usage may be reflected in two log records; these can be identified through the application IDs in the log records. For more information about governor log files, see “Governor Log Files” on page 278.

interval
The interval, in seconds, at which the daemon wakes up. If no interval is specified, an interval of 120 seconds is used.

You combine the following rule clauses to form a rule (that is, the full rule is followed by a semicolon, and not each individual clause). The clauses specify the time during which the rule applies, the limit on resource that can be used, and, optionally, specific users or applications and any action for the governor to take if a limit specified in the rule is exceeded. The clauses can only be specified once in a rule, but can be specified in more than one rule. The clauses must be specified in the order shown. In the description that follows, a [ ] indicates an optional clause.

[desc] Specifies a text description for the rule. The description must be enclosed by either single or double quotation marks.

[time] Specifies the time period during which the rule is to be evaluated.

The time period must be specified in the following format `time hh:mm hh:mm`, for example, `time 8:00 18:00`. If this clause is not specified, the rule is valid 24 hours a day.

[authid]
Specifies one or more authorization ids (authid) under which the application is executing. Multiple authids must be separated by a comma (,), for example `authid gene, michael, james`. If this clause does not appear in a rule, the rule applies to all authids.

[applname]
Specifies the name of the executable (or object file) that makes the connection to the database.

Multiple application names must be separated by a comma (,), for example, `applname db2bp, batch, geneprog`. If this clause does not appear in a rule, the rule applies to all application names.

Notes:

1. Application names are case sensitive.
2. The database manager truncates all application names to 20 characters. You should ensure that the application you want to govern is uniquely identified by the first 20 characters of its application name; otherwise, an unintended application may be governed.

   Application names specified in the governor configuration file are truncated to 20 characters to match their internal representation.

setlimit
Specifies one or more limits for the governor to check. The limits can only be -1 or greater than 0 (for example, `cpu -1 locks 1000 rowssel 10000`). At least one of the limits (cpu, locks, rowsread, uowtime) must be specified, and any limit not specified by the rule is not limited by that particular rule. The governor can check the following limits:

cpu *nnn*
Specifies the number of CPU seconds that can be consumed by an application. If you specify `-1`, the governor does not limit the application's CPU usage.

Note: This option is not available in Windows NT or OS/2 environments.

locks *nnn*
: Specifies the number of locks that an application can hold. If you specify `-1`, the governor does not limit the number of locks held by the application.

rowssel *nnn*
: Specifies the number of rows that are returned to the application. This value will only be non-zero at the coordinator node. If you specify `-1`, the governor does not limit the number of rows that can be selected.

uowtime *nnn*
: Specifies the number of seconds that can elapse from the time that a unit of work (UOW) first becomes active. If you specify `-1`, the elapsed time is not limited.

Note: If you used the `sqlmon` (Database System Monitor Switch) API to deactivate the unit of work switch, this will affect the ability of the governor to govern applications based on the unit of work elapsed time. The governor uses the monitor to collect information about the system. If you turn off the switches in the database manager configuration file, then it is turned off for the entire instance, and governor will no longer receive this information.

idle *nnn*
: Specifies the number of idle seconds allowed for a connection before a specified action is taken. If you specify `-1`, the connection's idle time is not limited.

rowsread *nnn*
: Specifies the number of rows an application can select. If you specify `-1`, there is no limit on the number of rows the application can select.

Note: This limit is not the same as rowssel. The difference is that rowsread is the count of the number of rows that had to be read in order to return the result set. The number of rows read includes reads of the catalog tables by the engine and may be diminished when indices are used.

[action]
: Specifies the action to take if one or more of the specified limits is exceeded. You can specify the following actions.

Note: If a limit is exceeded and the action clause is not specified, the governor reduces the priority of agents working for the application by 10.

priority *nnn*
Specifies a change to the priority of agents working for the application. Valid values are from −20 to +20.

For this parameter to be effective:
- On UNIX-based platforms, the *agentpri* database manager parameter must be set to the default value; otherwise, it overrides the priority clause.
- On OS/2 and Windows NT platforms, the *agentpri* database manager parameter and *priority* action may be used together.

force Specifies to force the agent that is servicing the application. (Issues a FORCE APPLICATION to terminate the coordinator agent.)

schedule [class]
Scheduling improves the priorities of the agents working on the applications with the goal of minimizing the average response times while maintaining fairness across all applications.

The governor enforces its schedule by setting priorities for the agents working on the applications, using query cost estimates from the DB2 internal query compiler. If the class option is specified, all applications chosen by the rule are scheduled among themselves only. If this option is not specified, the governor uses one or more classes, with scheduling done within each class.

Within each class, how an application is prioritized is based on:
- The number of locks held by the application within the class. (An application holding up many other applications due to locking is given a high priority.)
- The application's age. (An application in the system for a long time is given a high priority.)
- The application's estimated remaining running time. (An application close to finishing is given a high priority.)

Applications that are not covered by any schedule run with the highest authority.

Note: If you used the `sqlmon` (Database System Monitor Switch) API to deactivate the statement switch, this will affect the ability of the governor to govern applications based on the statement elapsed time. The governor uses the monitor to collect information about the system. If you turn off the switches in the database manager configuration file, then it is turned off for the entire instance, and governor will no longer receive this information.

The schedule action can:

- Ensure that applications in different groups each get time without all applications splitting time evenly.

  For instance, if 12 applications (three short, five medium, and six long) are running at the same time, they may all have poor response times because they are splitting the CPU. The database administrator can set up two groups, medium-length applications and long-length applications. Using priorities, the governor permits all the short applications to run, and ensures that at most three medium and three long applications run simultaneously. To achieve this, the governor configuration file contains one rule for medium-length applications, and another rule for long applications. The following example shows a portion of a governor configuration file that illustrates this point:

  ```
  desc "Group together medium applications in 1 schedule class"
  applname medq1, medq2, medq3, medq4, medq5
  setlimit cpu -1
  action schedule class;

  desc "Group together long applications in 1 schedule class"
  applname longq1, longq2, longq3, longq4, longq5, longq6
  setlimit cpu -1
  action schedule class;
  ```

- Ensure that each of several user groups (for example, organizational departments) gets equal prioritization.

  If one group is running a large number of applications, the administrator can ensure that other groups are still able to obtain reasonable response times for their applications. For instance, in a case involving three departments (Finance, Inventory, and Planning), all the Finance users could be put into one group, all the Inventory users could be put into a second, and all the Planning users could be put into a third group. The processing power would be split more or less evenly among the three departments. The following example shows a portion of a governor configuration file that illustrates this point:

```
desc "Group together Finance department users"
authid tom, dick, harry, mo, larry, curly
setlimit cpu -1
action schedule class;

desc "Group together Inventory department users"
authid pat, chris, jack, jill
setlimit cpu -1
action schedule class;

desc "Group together Planning department users"
authid tara, dianne, henrietta, maureen, linda, candy
setlimit cpu -1
action schedule class;
```

- Let the governor schedule all applications.

  If the class option is not included with the action, the governor creates its own classes based on how many applications fall under the schedule action, and puts applications into different classes based on the DB2 query compiler's cost estimate for the query the application is running. The administrator can choose to have all applications scheduled by not qualifying which applications are chosen. That is, no *applname* or *authid* clauses are supplied, and the *setlimit* clause causes no restrictions.

Note: If a limit is exceeded and the action clause is not specified, the governor reduces the priority of agents working for the application.

If more than one rule applies to an application, the rule that is closest to the end of the configuration file is applied to the application. An exception occurs if -1 is specified for a clause in a rule. In this situation, the value specified for the clause in the subsequent rule can only override the value previously specified for the *same* clause: other clauses in the previous rule are still operative. For example, one rule indicates that the priority of an application is to be decreased if its elapsed time is greater than 1 hour, or if it selects more than 100 000 rows (that is, `rowssel 100000 uowtime 3600`). A subsequent rule indicates that the same application can have unlimited elapsed time (that is, `uowtime -1`). In this situation, if the application runs for more than 1 hour, its priority won't be changed (that is, `uowtime -1` overrides `uowtime 3600`), but if it selects more than 100 000 rows, its priority will be lowered (as `rowssel 100000` is still valid).

Figure 29 on page 277 shows an example of a configuration file.

```
{ Wake up once a second, the database name is ibmsampl
 do accounting every 30 minutes.  }
interval 1; dbname ibmsampl; account 30;

desc "CPU restrictions apply 24 hours a day to everyone"
setlimit cpu 600 rowssel 1000000 rowsread 5000000;

desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;

desc 'Slow down a subset of applications'
applname jointA, jointB, jointC, quryA
setlimit cpu 3 locks 1000 rowssel 500 rowsread 5000;

desc "Have governor prioritize these 6 long apps in 1 class"
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
action schedule class;

desc "Schedule all applications run by the planning dept"
authid planid1, planid2, planid3, planid4, planid5
setlimit cpu -1
action schedule;

desc "Schedule all CPU hogs in one class which will control consumption"
setlimit cpu 3600
action schedule class;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;

desc "During day hours do not let anyone run for more than 10 seconds"
time 8:30 17:00 setlimit cpu 10 action force;

desc "Allow users doing performance tuning to run some of
     their applications during lunch hour"
time 12:00 13:00 authid ming, geoffrey, john, bill
applname tpcc1, tpcc2, tpcA, tpvG setlimit cpu 600 rowssel 120000 action force;

desc "Some people should not be limited -- database administrator
  and a few others.  As this is the last specification in the
  file, it will override what came before."
authid gene, hershel, janet setlimit cpu -1 locks -1 rowssel -1 uowtime -1;

desc "Increase the priority of an important application so it always
      completes quickly"
applname V1app setlimit cpu 1 locks 1 rowssel 1 action priority -20;
```

Figure 29. Example Governor Configuration File

Governor Log Files

When a governor daemon forces an application, reads the governor configuration file, changes an application's priority, encounters an error or warning, starts, or ends, it writes a record to a log file. A separate log file exists for each governor daemon. This prevents file-locking bottlenecks that would result from many governor daemons writing to the same file at the same time. You can use the `db2govlg` utility to merge the log files together and query them. This utility is described in "Querying Governor Log Files" on page 279.

The log files are stored in the `log` subdirectory of the `sqllib` directory. (On Windows NT, the `log` subdirectory is under the instance directory.) You provide the base name for the log file when you issue the `db2gov` command. You should ensure that the log file name contains the database name, because there will be a log file for each node of each database that is being governed. In a partitioned database environment, the node number of the database partition that the governor is running on is automatically appended to the log file name to ensure that the filename is unique for each governor.

Each record in the log file has the following format:

```
Date Time NodeNum RecType Message
```

The *Date* and *Time* field is in the yyyy-mm-dd-hh.mm.ss format, so that you can merge the log files for each database partition by sorting on this field.

The *NodeNum* field indicates the number of the database partition on which the governor is running.

The *RecType* field contains different values, depending on the type of log record being written to the log. The values that can be recorded are:
- `START` to indicate that the governor was started
- `FORCE` to indicate that an application was forced
- `PRIORITY` to indicate that the priority of an application was changed
- `ERROR` to indicate an error
- `WARNING` to indicate a warning
- `READCFG` to indicate that the governor read the configuration file
- `STOP` to indicate that the governor was stopped
- `ACCOUNT` to indicate the application's accounting statistics.

  The fields are:
  - `authid`
  - `appl_id`
  - `written_usr_cpu`
  - `written_sys_cpu`
  - `appl_con_time`

- `SCHEDULE` to indicate that a change in agent priorities occurred.

Because standard values are written, you can query the log files for different types of actions. The *Message* field provides other nonstandard information that varies according to the value under the *Rectype* field. For instance, a `FORCE` or `NICE` record indicates application information in the *Message* field, while an `ERROR` record includes an error message.

An example log file is as follows:

```
1995-12-11 14.54.52    0 START      Database = TQTEST
1995-12-11 14.54.52    0 READCFG    Config = /u/db2instance/sqllib/tqtest.cfg
1995-12-11 14.54.53    0 ERROR      SQLMON Error: SQLCode = -1032
1995-12-11 14.54.54    0 ERROR      SQLMONSZ Error: SQLCode = -1032
```

Querying Governor Log Files

Each governor daemon writes to its own log file. You can use `db2govlg` utility to query the log file. You can list the log files for a single partition, or for all database partitions, sorted by date and time. You can also query on the basis of the *RecType* log field. The syntax for `db2govlg` is as follows:

```
►►──db2govlg──log-file──┬──────────────────────┬──┬──────────────────────────┬──────────►◄
                        └─nodenum──node-num────┘  └─rectype──record-type─────┘
```

Figure 30. Syntax for db2govlg

The parameters are as follows:

log-file
: The base name of the log file (or files) that you want to query.

nodenum **node-num**
: The node number of the database partition on which the governor is running.

rectype **record-type**
: The type of record that you want to query. The record types are:
  - START
  - READCFG
  - STOP
  - FORCE
  - NICE
  - ERROR
  - WARNING
  - ACCOUNT

There are no authorization restrictions for using this utility. This allows all users to query whether the governor has affected their application. If you want to restrict access to this utility, you can change the group permissions for the `db2govlg` file.

Running the Governor and Database Manager Performance

The governor can affect database manager performance because it requests snapshots of the database manager. If the governor uses too much CPU, you can increase its wake-up interval to reduce its CPU usage.

Chapter 10. Scaling Your Configuration Through Adding Processors

You may find that the characteristics of your configuration are not appropriate for your current and planned needs. As a result, you should consider actions that would increase your configuration's capacity, performance, or both. For example, adding containers to your configuration increases your capacity to store data, but also can improve performance during utility use (such as when loading data). Other ways to improve capacity or performance include: adding memory and adding processors in either a symmetric multiprocessor or partitioned database environment.

The focus of this chapter is on improving performance by increasing the number of processors in your configuration.

You should consider scaling your configuration as discussed in the remainder of this chapter if:

- You had a single-partition configuration with a single processor that was being used to its maximum capacity. As a result, you have decided to change configurations and have:
  - Determined a symmetric multiprocessor (SMP) configuration is your best choice for a new environment. You perhaps made this choice because you want to take advantage of the processing power available with more than one processor. Each processor shares memory and storage system resources. All of the processors are within one system, so there are no additional considerations such as communication lines between systems, perhaps no additional administration staff to support any new systems, and coordination of tasks between systems is not an issue. DB2 Universal Database supports this environment.
  - Determined a partitioned database configuration is your best choice for a new environment. You perhaps made this choice because you want to take advantage of the processing power available with more than one processor that is physically separate from the first. Each processor has its own memory and storage system resources without having to share with the other processor. While you may have the additional considerations mentioned above (communications, staff, and coordination of tasks), there are advantages to this choice such as the ability to balance data and user access across more than one system. DB2 Universal Database supports this environment.
- You currently have a SMP configuration and you are planning to add one or more additional processors. In this case, you are already familiar with

those considerations associated with this type of environment. By adding one or more additional processors, you are simply adding computing power to your environment without adding new considerations. DB2 Universal Database supports this environment.

- You have a partitioned database configuration and you are planning to add one or more additional database partitions. In this case, you are already familiar with those considerations associated with this type of environment. By adding one or more additional database partitions, you are simply adding computing power to your environment without adding new considerations other than making the transition to the larger number of partitions. DB2 Universal Database supports this environment.

  A variation on the partitioned database configuration is one where the database partitions are SMP machines. DB2 Universal Database supports this environment.

When you scale your system by changing the environment, you should be aware of the impact that such a change can have on your database procedures such as loading data, backing up the database, and restoring the database.

When you add a new database partition, you cannot drop or create a database that takes advantage of the new partition until the procedure is complete, and the new server is successfully integrated into the system.

Adding Processors to a Machine

If the existing processors are fully utilized much of the time, consider installing one or more additional processors in your machine. To allow the DB2 database manager to take advantage of the new processors, there are configuration parameters that should be reviewed and perhaps updated. (Some operating systems, like Solaris, can dynamically vary processors on- and off-line.) The parameters that are used to determine the number of processors used and may need to be updated include:

- "Default Degree (dft_degree)" on page 421
- "Maximum Query Degree of Parallelism (max_querydegree)" on page 443
- "Enable Intra-Partition Parallelism (intra_parallel)" on page 445

You should also consider the parameters associated with applications that may need to be updated. See "Parallel Processing of Applications" on page 84 for more information.

When working in an environment where TCP/IP is used for communication, you should consider the value for the DB2TCPCONNMGRS registry variable. See "Appendix A. DB2 Registry and Environment Variables" on page 469 for more information on this variable.

Adding Database Partitions to a Partitioned Database System

You can add database partitions to the partitioned database system either when it is running, or when it is stopped. The following sections describe how to do this task. Because adding a new server can be time consuming, you may want to do it when the database manager is already running. The procedure is described in “Adding Database Partitions to a Running System” on page 284.

The ADD NODE command is used to add a database partition to a system. This command can be invoked:

- As an option on `db2start`
- Using:
  - The command line processor ADD NODE command
  - `sqleaddn`
  - `sqlepstart`.

The method you use to invoke the command is dependent upon whether your system is stopped (using `db2start`) or running (using any of the other choices).

When a new database partition is added to the system using the ADD NODE command, all existing databases in the instance are created on the new database partition. You can also specify which containers for temporary table spaces will be used with the databases that are created. The containers can be:

- The same as those defined for the catalog node for each database. (This is the default.)
- The same as those defined for another database partition.
- Not created at all. The ALTER TABLESPACE statement must be used to add temporary table space containers to each database before the database can be used.

A database on the new partition cannot be used to contain data until one or more nodegroups are altered to include the new database partition. See “Adding and Dropping Database Partitions” on page 292 for more information on how to alter a nodegroup.

Note: If there are no databases defined in the system and you are running DB2 Enterprise - Extended Edition on a UNIX-based system, edit the `db2nodes.cfg` file to add a new database partition definition; do not use any of the following procedures, as they apply only when a database exists. Refer to “Altering a Nodegroup” in the *Administration Guide: Planning* for more information on how to update the node configuration file.

Windows NT Considerations: If you are using DB2 Enterprise - Extended Edition on Windows NT and have no databases in the instance, you should use the DB2NCRT command to scale the database system. For information about this command, refer to the *Command Reference*. If, on the other hand, you already have databases, you should use the DB2START ADDNODE command, as this ensures that a database partition is created for each existing database when you scale the system. For information about the DB2START command and the parameters that you must use on Windows NT, refer to the *Command Reference*. On Windows NT, you should never manually edit the node configuration file (`db2nodes.cfg`), as this can introduce inconsistencies into the file.

Adding Database Partitions to a Running System

You can add new database partitions to a partitioned database system while it is running and while applications are connected to databases. However, a newly added server does not become available to all databases until the database manager is shut down and restarted.

To add a database partition to a multiple server system:

1. If the database partition is to be created on a server that already exists in the system, go to the next step. Otherwise, do the following:
   - On UNIX platforms,
     a. Install the new server. This includes making executables accessible (using shared file-system mounts or local copies), synchronizing operating system files with those on existing processors, ensuring that the `sqllib` directory is accessible as a shared file system, and ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values.
     b. Register the host name with the name server or in the `hosts` file in the `etc` directory on all database partitions.
   - On Windows NT platforms,
     a. Install the new server.
     b. Run the ADD NODE command on the new server. This command causes a database partition to be created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it.
     c. Go to point three (3).
2. Run the DB2START command on any database partition, specifying the NODENUM, ADDNODE, HOSTNAME, PORT, and NETNAME parameters. On the Windows NT platform, you must also specify the COMPUTER, USER, and PASSWORD parameters. For more information about the DB2START command, refer to the *Command Reference*.

You can also optionally specify the source for any temporary table space container definitions that need to be created with the databases. If no table space information is provided, the temporary table space container definitions are retrieved from the catalog node for each database.

When the command completes, the new server is stopped. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD NODE command (which is called when the ADDNODE parameter is specified) runs on the correct database partition. When the utility ends, the new server is stopped.

3. Stop the database manager by running the DB2STOP command.

   When you stop all the database partitions in the system, the node configuration file is updated to include the new database partition.
4. Start the database manager by running the DB2START command.

   The newly added database partition is now started along with the rest of the system.

   When all the database partitions in the system are running, system-wide activities, such as creating or dropping a database, can be done.

   Note: You may have to issue the DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
5. Optionally, take a backup of all databases on the new database partition.
6. Optionally, redistribute data to the new database partition. For details, see "Chapter 11. Redistributing Data Across Database Partitions" on page 291.

Adding Database Partitions to a Stopped System

You can add new database partition to a partitioned database system while it is stopped. The newly added server becomes available to all databases when the database manager is started up again. You have two options. You can either have the database manager update the node configuration file for you, or you can do it manually. The preliminary steps for both procedures are the same.

Note: You should not update the node configuration file manually while working on Windows NT. Instead, you should use the database manager to update this file (as described below).

To add a new database partition to a multiple server system:
1. Issue DB2STOP to stop all the database partitions.
2. If the server is to be created on a processor that already exists in the system, go to the next step. Otherwise, do the following:
   a. On UNIX platforms,
      1) Install the new server. This includes making executables accessible (using shared file-system mounts or local copies), synchronizing

operating system files with those on existing processors, ensuring that the `sqllib` directory is accessible as a shared file system, and ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values.

   2) Register the host name with the name server or in the `hosts` file in the `etc` directory on all database partitions.

b. On Windows NT platforms,

   1) Install the new server.

   2) Run the ADD NODE command on the new server. This command causes a database partition to be created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it.

   3) Run the DB2START command to start the database system. Note that the node configuration file (`db2nodes.cfg`) has already been updated to include the new server during the installation of the new server.

   4) Optionally redistribute data onto the new server. See "Chapter 11. Redistributing Data Across Database Partitions" on page 291 for more details on how to do this.

c. If you want the database manager to update the `db2nodes.cfg` file for you, continue with the instructions in "Having the Database Manager Update the Node Configuration File".

   Note: On Windows NT, you should not edit the `db2nodes.cfg` file manually, as this can introduce inconsistencies into the file. Instead, you should have the database manager update this file.

   If you want to update the `db2nodes.cfg` file yourself, continue with the instructions in "Updating the Node Configuration File Manually" on page 287.

Having the Database Manager Update the Node Configuration File

Following your adding of one or more new database partitions to your partitioned database system, to complete making the new partition available you must update the `db2nodes.cfg` file. In point "c" above, you made the decision to have the database manager update the node configuration file. How this is done is presented in this section.

Note: If, in point "c" above, you made the decision to update the node configuration file manually, then you should skip this section and go to the next.

Continue the procedure as follows:

1. Run the DB2START command on the new database partition specifying NODENUM, ADDNODE, HOSTNAME, PORT, and NETNAME parameters. On the Windows NT platform, you must also specify the COMPUTER, USER, and PASSWORD parameters. For more information about the DB2START command, refer to the *Command Reference*. The values that you specify for these parameters are used to update the node configuration file.

   When the command completes, the new server is stopped. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD NODE command (which is called when the ADDNODE parameter is specified) runs on the correct database partition. When the utility ends, the new server is stopped.
2. Issue the DB2STOP command.

   When you issue the DB2STOP command, the node configuration file is updated to include the new server.
3. Issue the DB2START command to start the database system.

   Note: You may have to issue the DB2START command twice for all database partition servers to access the new node configuration file.
4. Optionally, take a backup of all databases on the new database partition.
5. Optionally, redistribute data to the new server. For details, see "Chapter 11. Redistributing Data Across Database Partitions" on page 291.

Updating the Node Configuration File Manually

Following your adding of one or more new database partitions to your partitioned database system, to complete making the new partition available you must update the `db2nodes.cfg` file. In point "c" before the previous section, you made the decision to update the node configuration file manually. (Recall that you should not manually update the node configuration file when working on Windows NT.) How to update the node configuration file manually is presented in this section.

Note: If, in point "c" before the previous section, you made the decision to have the database manager update the node configuration file, then you should go back to the previous section.

Continue the procedure as follows:

1. Edit the `db2nodes.cfg` file and add the new database partition to it.
2. Issue the following command to start the new node:

   ```
   DB2START NODENUM nodenum
   ```

   Specify the number you are assigning to the new database partitioned server as the value of *nodenum*.

3. If the new server is to be a logical database partition (that is, it is not node 0), use **db2set** command to update the DB2NODE registry value, specifying the number of the server you are adding.
4. Run the ADD NODE command on the new server.

   This command also causes a database partition to be created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it.
5. When the ADD NODE command completes, issue the DB2START command to start the other database partitions in the system.

   You should not attempt to do any system-wide activities, such as creating or dropping a database, until all database partitions are successfully started.
6. Optionally, take a backup of all new database partitions on the new server.
7. Optionally, redistribute data to the new database partition. For details, see "Chapter 11. Redistributing Data Across Database Partitions" on page 291.

Dropping a Database Partition from a System

You can drop a database partition by using the DB2STOP command with the DROP NODENUM parameter, or the `sqlepstp` API. Before doing this, you must first ensure that the database partition being dropped is not being used by any database. To check, issue the DROP NODE VERIFY command.

You should ensure that all transactions for which this database partition was the coordinator have all committed or rolled back successfully. This may require doing crash recovery on other servers.

For example, if you drop the coordinator database partition (that is, the coordinator node), and another database partition participating in a transaction crashed before the coordinator node was dropped, the crashed database partition will not be able to query the coordinator node for the outcome of any indoubt transactions.

To drop a database partition from a partitioned database system:
1. Redistribute the data for every database that resides on this node. This satisfies the requirement that the database partition being dropped is not being used by any database. For details, see "Chapter 11. Redistributing Data Across Database Partitions" on page 291.
2. Issue the DROP NODE VERIFY command or the `sqledrpn` API to verify that the server is not in use.

Depending on the message you receive, proceed with either step 3 or step 4.

3. If you receive message SQL6034W (Node not used in any database), you can do the following:
    a. Issue the DB2STOP command with the DROP NODENUM parameter to drop the database partition. After the command completes successfully, the system is stopped.
    b. If you want to, start the database manager with the DB2START command.
4. If you receive message SQL6035W (Node in use by database), do the following:
    a. Use the REDISTRIBUTE NODEGROUP command to redistribute the data from the database partition you are dropping to other database partitions from the database alias, as indicated in message SQL6035W. You cannot drop the database partition until this is done.
    b. Drop any event monitors defined on the database partition.
    c. Return to step 2 on page 288 and continue.

Chapter 11. Redistributing Data Across Database Partitions

Only if you are working in a partitioned database environment do you need to be concerned with redistribution of data. If you are in a single partition database environment there is no need for you to use the information found here.

You use the Data Redistribution utility to move data among the database partitions in an existing nodegroup. You can use it to do the following:

- Balance data volumes and processing loads across database partitions.

  This is useful if you have a database table in which all the data is accessed on a regular basis.
- Introduce skew in the data distribution across database partitions.

  This is useful if you have a database table in which only some of the data is accessed on a regular basis. In this situation, you could redistribute the table so that the infrequently accessed data is on a small number of database partitions in the nodegroup, and the frequently accessed data is distributed over a larger number of partitions. This would improve access performance and throughput on the most frequently run applications.

The REDISTRIBUTE NODEGROUP command is how you invoke the Data Redistribution utility. Refer to the *Command Reference* for details on the syntax for this command.

To preserve table collocation, this operation is applied to all tables in a nodegroup, and redistribution is done at the nodegroup level rather than at the table level.

To achieve the data distribution that you want, the utility uses a partitioning map to move the rows of the tables among the database partitions of the nodegroup. Depending on the option you specify, the utility can generate a target partitioning map or can use an existing partitioning map as input.

Notes:

1. You should specify a log file size based on the log space requirements you think that the Data Redistribution operation will need. You should also ensure that the log is large enough to accommodate the INSERT and DELETE operations done at each database partition where data is being redistributed.
2. If you want to redistribute the data in a nodegroup that contains replicated summary tables, you must first drop these tables, redistribute

the nodegroup, then re-create the tables. You cannot redistribute a nodegroup that contains replicated summary tables.

How to Partition Data

By default, the Data Redistribution utility assumes that the same number of rows hash to each hash partition, therefore it partitions the hash partitions uniformly across all the database partitions of the nodegroup. If the same number of rows do not hash to each hash partition, you can use a *distribution file* to specify the current distribution. This file contains a value for each of the 4 096 hash partitions. Each value is used as the *weight* of the corresponding hash partition. The Data Redistribution utility generates a target partitioning map in which all the database partitions have about the same weight. Thus, the distribution file can be used to achieve uniform data distribution even if the data distribution is skewed.

The `AutoLoader` utility can be used to create a data distribution file using the ANALYZE option. You can use this file as input to the Data Redistribution utility. Refer to the *Data Movement Utilities Guide and Reference* for more information on the `AutoLoader` utility.

Alternatively, you can use the PARTITION and NODENUMBER SQL functions to determine the current data distribution across hash partitions or database partitions. (You use the PARTITION function to determine the distribution across hash partitions.) You can use this information to derive both a distribution file and a target partitioning map.

For example, to see which database partitions, if any, have an atypically large number of rows due to non-uniform data distribution:

```
SELECT PARTITION(column_name), COUNT(*) FROM table_name
   GROUP BY PARTITION(column_name)
   ORDER BY PARTITION(column_name) DESC
   FETCH FIRST 100 ROWS ONLY
```

You should ensure that `table_name` is the largest table and `column_name` is an appropriate column from that table.

Adding and Dropping Database Partitions

You can use the ALTER NODEGROUP statement to add or drop database partitions from a nodegroup. When adding database partitions, the partitions must already be defined in the node configuration file.

Following the use of the ALTER NODEGROUP statement, a new partitioning map is created. This new partitioning map can become the target partitioning map when using the Data Redistribution utility. (The other way to create the target partitioning map is to create it yourself.)

If you use the ALTER NODEGROUP statement with the WITHOUT TABLESPACES clause, you must add table space containers to a new database partition (or partitions) before redistributing the data. For additional information about the ALTER NODEGROUP statement, refer to the *SQL Reference*.

Specifying a Target Partitioning Map

The Data Redistribution utility uses a partitioning map to do the data redistribution. It can create its own target partitioning map, or you can provide one for the utility to use. If you create one, the entry or entries determine the type of nodegroup that results from the data redistribution:
- 1 entry for a single-partition nodegroup
- 4 096 entries for a multipartition nodegroup

If the target partitioning map has more than one database partition, all tables in the nodegroup must have the same partitioning key defined.

The target partitioning map can only contain database partition numbers that are defined in the SYSCAT.NODEGROUPDEF catalog table, excluding those with an IN_USE value of ’T’. (’T’ means that the partition is not in the target partitioning map.) All database partitions that have an IN_USE value of ’D’ (meaning to drop) and do not appear in the target partitioning map are dropped when the redistribution operation has completed successfully.

How Data Is Redistributed Across Database Partitions

The Data Redistribution operation is done on the set of tables in the specified nodegroup of a database. (The application must be connected to the database at the catalog database partition before executing the operation.) The utility uses both the source partitioning map and the target partitioning map to identify which hash partitions have been assigned to a new location (that is, a new database partition number). All rows that correspond to a partition that has a new location are moved from the database partition specified in the source partitioning map to the database partition specified in the target partitioning map.

The Data Redistribution utility does the following:
1. Obtains a new partitioning map ID for the target partitioning map, and inserts it into the SYSCAT.PARTITIONMAPS catalog view.

2. Updates the REBALANCE_PMAP_ID column in the SYSCAT.NODEGROUPS catalog view for the nodegroup with the new partitioning map ID.
3. Adds any new database partitions to the SYSCAT.NODEGROUPDEF catalog view.
4. Sets the IN_USE column in the SYSCAT.NODEGROUPDEF catalog view to ’D’ for any database partition that is to be dropped.
5. Does a COMMIT for the catalog updates.
6. Creates database files for all new database partitions.
7. Redistributes the data on a table-by-table basis for every table in the nodegroup. This is described in “How Data Is Redistributed in Tables”.
8. Deletes database files and deletes entries in the SYSCAT.NODEGROUPDEF catalog view for database partitions that were previously marked to be dropped.
9. Updates the nodegroup record in the SYSCAT.NODEGROUPS catalog view to set PMAP_ID to the value of REBALANCE_PMAP_ID and REBALANCE_PMAP_ID to NULL.
10. Deletes the old partitioning map from the SYSCAT.PARTITIONMAPS catalog view.
11. Does a COMMIT for all changes.

How Data Is Redistributed in Tables

When doing data redistribution on a table, the utility does the following:

1. Locks the row for the table in the SYSTABLES catalog table.
2. Invalidates all packages that involve this table. The partitioning map ID associated with the table will change because the table is being redistributed. Because the packages are invalidated, the compiler must obtain the new partitioning information for the table and generate packages accordingly.
3. Locks the table in exclusive mode.
4. Redistributes the data in the table via DELETEs and INSERTs.
5. If the redistribution operation succeeds, it:
   a. Issues a COMMIT for the table.
   b. Continues with the next table in the nodegroup.

   If the operation fails before the table is fully redistributed, the utility:
   a. Issues a ROLLBACK on updates to the table.
   b. Ends the entire redistribution operation and returns an error.

Estimating the log space requirements when distributing data is important. The log must be large enough to accommodate the INSERT and DELETE operations at each database partition where data is being redistributed. The heaviest logging requirements will be either on the database partition that will lose the most data, or on the database partition that will gain the most data. If you are moving to a larger number of database partitions, then the ratio of current database partitions to the new number of database partitions will assist in determining the number of INSERT and DELETE operations.

For example, if you are moving from four to five database partitions, approximately twenty percent of the four original database partitions will have data moved to the new database partition. This means that the four original database partitions will each experience twenty percent DELETE operations based on the total amount of the data at each database partition. The new database partition will experience all of the INSERT operations (that is, the equivalent of an equal number of the DELETE operations from all of the four original database partitions).

The above example assumes a uniform distribution of the data. There may also be a case where there is a non-uniform distribution of the data as in the case where there is a large number of NULL values in the partitioning key. In this case, all of these rows would end up on one database partition under the old partitioning scheme and on a different database partition under the new partitioning scheme. As a result, this can increase the amount of log space required on those two database partitions perhaps well beyond the amount calculated by assuming uniform distribution.

When doing the actual calculations, you must multiply the percentage of change (like twenty percent) by the size of the largest table. You do this because the redistribution of each table is accomplished as a single transaction.

Note: However, the largest table may be uniformly distributed but the second largest table (for example) may have one or more inflated database partitions. In such a case, you should consider using the second table and not the largest one.

Once you have calculated the maximum amount of data to be inserted and deleted at a database partition, double that figure to determine the peak size of the active log. If this exceeds the active log limit of 32 GB, then the data redistribution must be done in steps. There is a utility called “makepmap” that can be used to generate a series of target partition maps, one for each step.

Recovering From Redistribution Errors

After the redistribution operation begins to execute, a file is written to the `redist` subdirectory of the `sqllib` directory. This status file lists any operations that are done on database partitions, the names of the tables that were redistributed, and the completion status of the operation. If a table cannot be redistributed, its name and the applicable SQLCODE is listed in the file. If the redistribution operation cannot begin because of an incorrect input parameter, the file is not written and an SQLCODE is returned.

The file has the following naming convention:

```
databasename.nodegroupname.timestamp   (for UNIX platforms)
databasename\nodegroupname\date\time   (for non-UNIX platforms)
```

Note: On non-UNIX platforms, only the first eight (8) bytes of the `nodegroupname` are used.

If the data redistribution operation fails, some tables may be redistributed, while others are not. This occurs because data redistribution is performed a table at a time. You have two options for recovery:

- Use the CONTINUE option to continue the operation to redistribute the remaining tables.
- Use the ROLLBACK option to undo the redistribution and set the redistributed tables back to their original state. The rollback operation can take about the same amount of time as the original redistribution operation.

Before you can use either option, a previous data redistribution operation must have failed such that the REBALANCE_PMID column in the SYSNODEGROUPS catalog table is set to a non-NULL value.

If you happen to delete the status file by mistake, you can still attempt a CONTINUE operation.

Data Redistribution and Other Operations

You can do the following operations on objects of the nodegroup while the utility is running. You cannot, however, do them on the table that is being redistributed. You can:

- Create indexes on other tables. The CREATE INDEX statement uses the partitioning map of the affected table.
- Drop other tables. The DROP TABLE statement uses the partitioning map of the affected table.
- Drop indexes on other tables. The DROP INDEX statement uses the partitioning map of the affected table.
- Query other tables.

- Update other tables.
- Create new tables in a table space defined in the nodegroup. The CREATE TABLE statement uses the target partitioning map.
- Create table spaces in the nodegroup.

You cannot do the following operations while the utility is running:
- Another redistribution operation on the nodegroup
- An ALTER TABLE on any table in the nodegroup
- Drop the nodegroup
- Alter the nodegroup.

Following Data Redistribution

After completing the redistribution of data across a nodegroup, it is strongly recommended that you do a RUNSTATS to update the statistics associated with the tables that may have been redistributed.

For more information on the RUNSTATS command, refer to the *Command Reference* manual.

Chapter 12. Benchmark Testing

Benchmarking is a normal part of the application development life cycle. It is a team effort involving both application developers and database administrators (DBAs), and should be performed against your application in order to determine and improve performance. Assuming that the application code has been written as efficiently as possible, additional performance gains can be realized from tuning the database and database manager configuration parameters, and even you application parameters to meet the requirements of the application.

There are several different types of benchmarking. A *transaction per second* benchmark would determine the throughput capabilities of the database manager under certain limited laboratory conditions. An *application* benchmark would test the same throughput capabilities, but under conditions that are closer to those under which your application will run when it is implemented. Benchmarking for the purpose of tuning configuration parameters is based upon these "real-world" conditions, and involves repeatedly running SQL taken from your application with varying parameter values until your application runs as efficiently as possible.

The benchmarking methods described in this section are oriented towards the configuration parameters. However, the same basic technique can be used for tuning other factors that affect performance, such as:
- SQL statements
- Indexes
- Table space configuration
- Application code
- Hardware configuration.

Benchmarking is helpful in understanding how the database manager responds under varying conditions. You could create scenarios that test deadlock handling, utility performance, different methods of loading data, transaction rate characteristics as more users are added, and even the effect on the application of using a new release of the product.

The following topics are provided:
- "Benchmark Testing Methodology" on page 300
- "Preparing for Benchmark Testing" on page 300
- "Creating a Benchmark Program" on page 302
- "Executing the Benchmark Tests" on page 307.

Benchmark Testing Methodology

This benchmarking technique is based on the scientific method. A repeatable environment will be created in which the same test, run under the same conditions, will yield comparable results.

Benchmarking can also begin by running the test application in a normal environment. As a performance problem is narrowed down, specialized test cases can be developed to limit the scope of the function that is being tested and observed. The specialized test cases need not emulate an entire application in order to obtain valuable information. Start with simple measurements, and increase the complexity only when warranted.

Characteristics of good benchmarks (or measurements) include:
- Each test is repeatable.
- Each iteration of a test is started in the same system state.
- There are no functions or applications active in the system other than those being measured (unless the scenario includes some amount of other activity going on in the system).

  Note: Applications that are started use memory even when they are minimized or idle. This increases the likelihood of paging skewing the results of the benchmark and violating the repeatability rule.
- The hardware and software used for benchmarking matches your production environment.

As with any benchmarking, a scenario must be devised and then executed. The following information applies these concepts to the DB2 environment.
- "Preparing for Benchmark Testing"
- "Creating a Benchmark Program" on page 302
- "Executing the Benchmark Tests" on page 307.

Preparing for Benchmark Testing

The logical design of your application's database should be complete before performance benchmarking is started. Tables, views, and indexes need to be set up and populated. Tables should be normalized, application packages bound, and tables populated with realistic data.

You should have determined the final physical design of the database. The database manager objects should be placed in their final disk locations, log files sized, work files and backup locations determined, and backup procedures tested. In addition, packages should be checked to make sure that performance options such as row blocking are enabled when possible.

You should have reached a point in the application's programming and testing phases that will enable you to create your benchmark programs (see next section). An application's practical limits may be revealed during the benchmark testing; however, the purpose of the benchmark described here is to measure performance, not to detect defects or abends.

Your benchmarking test program will need to run in as accurate a representation of the final production environment as possible; ideally, on the same model of server with the same memory and disk configurations. This is especially important when the application will ultimately involve large numbers of users and large amounts of data. The operating system itself and any communications or file-serving facilities used directly by the benchmark should also have been tuned.

It is also important to benchmark with a production-size database. An individual SQL statement should return as much data and involve as much sorting as it will once it is implemented in production. Adhering to this rule will ensure that the application will incur representative memory requirements.

The type of SQL statements to be benchmarked should be either *representative* or *worst-case*, as described below:

Representative SQL

Representative SQL includes those statements that are executed during typical operations of the application being benchmarked. The statements that are selected will depend on the nature of the application. For example, a data-entry application might test an INSERT statement, while a banking transaction might test a FETCH, an UPDATE, and several INSERTs. The frequency of execution and volume of data processed by the statements chosen should be considered average. If the volumes are excessive, the statements should be considered under the *worst-case* category, even if they are typical SQL statements.

Worst-case SQL

Statements falling in this category include:

- Statements that are executed frequently.
- Statements that have high volumes of data being processed.
- Statements that are time-critical.

  For example, an application that is run when a telephone call is received from a customer and the statements must be run to retrieve and update the customer's information while the customer is waiting.
- Statements with the largest number of tables being joined or with the most complex SQL in the application.

For example, a banking application that produces combined customer statements of monthly activity for all their different types of accounts. A common table may list customer address and account numbers; however, several other tables must be joined to process and integrate all of the necessary account transaction information. Multiply the work necessary for one account by the several thousand accounts that must be processed during the same period, and the potential time savings drives the performance requirements.

- Statements that have a poor access path, such as one that is not executed very often and is not supported by the indexes that have been created for the table(s) involved.
- Statements that have a long elapsed time.
- A statement that is only executed at application initialization but has disproportionate resource requirements.

  For example, an application that generates a list of account work that must be processed during the day. When the application is started, the first major SQL statement causes a 7-way join, which creates a very large list of all the accounts for which this application user is responsible. The statement might only be run a few times per day, but takes several minutes to run when it has not been tuned properly.

Creating a Benchmark Program

There are a variety of factors to consider when designing and implementing a benchmark program. Since the main purpose of the program is to simulate a user application, the overall structure of the program can vary. You can use the entire application as the benchmark and simply introduce a means for timing the SQL statements to be analyzed. For large or complex applications, it may be more practical to just include blocks containing the important statements.

To test the performance of specific SQL statements, another approach would be to include these statements alone in the benchmark program along with the necessary CONNECT, PREPARE, OPEN, and other statements and a timing mechanism.

Another factor to consider is the type of benchmark to use. One option is to run a set of SQL statements repeatedly over a time interval. The ratio of the number of statements executed and this time interval would give the throughput for the application. Another option would be to simply determine the time required to execute individual SQL statements.

Regardless of the type of benchmark program, an efficient timing system is necessary to calculate the elapsed time, whether for individual SQL statements or the application as a whole. For simulating applications in which individual SQL statements would be executed in isolation, it may be important to consider times for CONNECT, PREPARE, and COMMIT statements. However, for programs processing many different statements, perhaps only a single CONNECT or COMMIT is necessary, so focusing on just the execution time for an individual statement may be the priority.

While the elapsed time for each query is an important factor in performance analysis, it may not necessarily reveal bottlenecks. For example, information on CPU usage, locking, and buffer pool I/O could show that the application is I/O bound instead of using the CPU to its full capacity. A benchmark program should allow you to obtain this kind of data for a more detailed analysis if needed.

Not all applications will need to send the entire set of rows retrieved from a query to some output device. For example, some may use the whole answer set as input for another program (that is, none of the rows are sent to output). Formatting data for screen output usually has high CPU cost and may not reflect user need. In order to provide an accurate simulation, a benchmark program should reflect the row handling of the specific application. If rows do get sent to an output device, inefficient formatting could consume the majority of CPU processing time and misrepresent the actual performance of the SQL statement itself.

The db2batch Benchmark Tool: A benchmark tool (`db2batch`) is provided in the `bin` subdirectory of your instance `sqllib` directory. This tool takes many of the points made above regarding the creating of a benchmark program into consideration. This tool will read SQL statements from either a flat file or standard input, dynamically describe and prepare the statements, and return an answer set. It also provides the added flexibility of allowing you to control the size of the answer set, as well as the number of rows that should be sent from this answer set to an output device.

You can also specify the level of performance-related information supplied, including the elapsed time, CPU and buffer pool usage, locking, and other statistics collected from the database monitor. If you are timing a set of SQL statements, `db2batch` will also summarize the performance results and provide both arithmetic and geometric means. For more information on invocation syntax, and options, type `db2batch -h` on a command line.

The *Command Reference* manual can also be referenced for more information on `db2batch`.

The following is an example of how `db2batch` could be used with an input file `db2batch.sql`:

```
-- db2batch.sql
-- ------------
--#SET PERF_DETAIL 3 ROWS_OUT 5

-- This query lists employees, the name of their department
-- and the number of activities to which they are assigned for
-- employees who are assigned to more than one activity less than
-- full-time.
--#COMMENT Query 1
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2;
--#SET PERF_DETAIL 1 ROWS_OUT 5
--#COMMENT Query 2
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2;
```

Figure 31. Sample Benchmark Input File: db2batch.sql

Using the following invocation of the benchmark tool:

```
   db2batch -d sample -f db2batch.sql
```

Produces the following output:

```
--#SET PERF_DETAIL 3 ROWS_OUT 5
Query 1

Statement number: 1

select lastname, firstnme,
deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2
```

Figure 32. Sample Output From db2batch (Part 1)

```
LASTNAME         FIRSTNME      DEPTNAME                      NUM_ACT
---------------------------------------------------------------------------
JEFFERSON        JAMES         ADMINISTRATION SYSTEMS                  3
JOHNSON          SYBIL         ADMINISTRATION SYSTEMS                  4
NICHOLLS         HEATHER       INFORMATION CENTER                      4
PEREZ            MARIA         ADMINISTRATION SYSTEMS                  4
SMITH            DANIEL        ADMINISTRATION SYSTEMS                  7
Number of rows retrieved is:       5
Number of rows sent to output is:   5
Elapsed Time is:           0.074      seconds
Locks held currently                         = 0
Lock escalations                             = 0
Total sorts                                  = 5
Total sort time (ms)                         = 0
Sort overflows                               = 0
Buffer pool data logical reads               = 13
Buffer pool data physical reads              = 5
Buffer pool data writes                      = 0
Buffer pool index logical reads              = 3
Buffer pool index physical reads             = 0
Buffer pool index writes                     = 0
Total buffer pool read time (ms)             = 23
Total buffer pool write time (ms)            = 0
Asynchronous pool data page reads            = 0
Asynchronous pool data page writes           = 0
Asynchronous pool index page reads           = 0
Asynchronous pool index page writes          = 0
Total elapsed asynchronous read time         = 0
Total elapsed asynchronous write time        = 0
Asynchronous read requests                   = 0
LSN Gap cleaner triggers                     = 0
Dirty page steal cleaner triggers            = 0
Dirty page threshold cleaner triggers        = 0
Direct reads                                 = 8
Direct writes                                = 0
Direct read requests                         = 4
Direct write requests                        = 0
Direct read elapsed time (ms)                = 0
Direct write elapsed time (ms)               = 0
Rows selected                                = 5
Log pages read                               = 0
Log pages written                            = 0
Catalog cache lookups                        = 3
Catalog cache inserts                        = 3
Buffer pool data pages copied to ext storage    = 0
Buffer pool index pages copied to ext storage   = 0
Buffer pool data pages copied from ext storage  = 0
Buffer pool index pages copied from ext storage = 0
Total Agent CPU Time (seconds)               = 0.02
Post threshold sorts                         = 0
Piped sorts requested                        = 5
Piped sorts accepted                         = 5
```

Figure 33. Sample Output From db2batch (Part 1)

```
--#SET PERF_DETAIL 1 ROWS_OUT 5
Query 2
Statement number: 2
select lastname, firstnme,
deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2
LASTNAME         FIRSTNME      DEPTNAME                      NUM_ACT
---------------------------------------------------------------------
GEYER            JOHN          SUPPORT SERVICES                      2
GOUNOT           JASON         SOFTWARE SUPPORT                      2
HAAS             CHRISTINE     SPIFFY COMPUTER SERVICE DIV.          2
JONES            WILLIAM       MANUFACTURING SYSTEMS                 2
KWAN             SALLY         INFORMATION CENTER                    2
Number of rows retrieved is:       8
Number of rows sent to output is:   5
Elapsed Time is:          0.037      seconds
Summary of Results
==================
                Elapsed            Agent CPU         Rows      Rows
Statement #     Time (s)           Time (s)          Fetched   Printed
1                     0.074              0.020         5         5
2                     0.037      Not Collected         8         5
Arith. mean     0.055
Geom.  mean     0.052
```

Figure 34. Sample Output from db2batch (Part 2)

The above sample output includes specific data elements returned by the database system monitor. For more information about these and other monitor elements, see the *System Monitor Guide and Reference* manual.

In the next example (on UNIX), just the summary table is produced.

```
db2batch -d sample -f db2batch.sql -r /dev/null,
```

Produces just the summary table. Using the `-r` option, `outfile1` was replaced by `/dev/null` and `outfile2` (which contains just the summary table) is empty, so `db2batch` sends the output to the screen:

```
Summary of Results
==================
                Elapsed             Agent CPU         Rows      Rows
Statement #     Time (s)            Time (s)          Fetched   Printed
1                     0.074               0.020         5         5
2                     0.037       Not Collected         8         5
Arith. mean     0.055
Geom.  mean     0.052
```

Figure 35. Sample Output from db2batch -- Summary Table Only

This benchmarking tool also has a CLI option. With this option, you can specify a cache size. In the following example, `db2batch` is run in CLI mode with a cache size of 30 statements:

```
db2batch -d sample -f db2batch.sql -cli 30
```

Executing the Benchmark Tests

One type of database benchmark involves choosing a configuration parameter and running the test with different values for that parameter until the maximum benefit is achieved. A single test should include executing the application through several iterations (for example, 10 times) with the same parameter value to get an average timing, which will better show the effect of parameter changes.

When running your benchmark, the first iteration (called a warm-up run) should be considered a separate case from the subsequent iterations (called normal runs). This is necessary because the results from the warm-up run will include some start-up activities (such as initializing the buffer pool). Consequently, the warm-up run will take somewhat longer than normal runs. Although the information from the warm-up run *may be* realistically valid, it will not be statistically valid. Therefore, when calculating the average timing or CPU for a specific set of parameter values, use the results from normal runs.

You may want to consider using the Performance Configuration wizard to create the warm-up run of the benchmark. The questions asked as part of the Performance Configuration wizard will provide insight into some of those things to consider when adjusting the configuration of your environment for the normal runs during your benchmark activity. To use the Performance Configuration wizard, enter `db2cc` to get into the Control Center and proceed from there.

If you are benchmarking using individual queries, you need to ensure that you minimize the potential effects of previous queries. This can be accomplished by flushing the buffer pool which can be done by reading a number of pages (irrelevant to your query) to fill the buffer pool.

After completing the iterations for a single set of parameter values, a single parameter can be changed. However, between each iteration, the following tasks should be performed to restore the benchmark environment to its original state:

- Return the application data and database manager statistics to their original state. If the catalog statistics were updated for the test, ensure the same values for the statistics are used for every iteration. The data used in the tests must be consistent if it is updated in the course of the tests. This can be done by:
  - Using the RESTORE utility to restore the entire database. The backup copy of the database would be in its previous state, and ready for the next test.
  - Using the IMPORT or LOAD utility to restore an exported copy of the data. This method allows you to restore only the data that has been affected. REORG and RUNSTATS utilities should be run against the tables and indexes containing this data.
- Return the application to its original state by re-BINDing it to the database.

  The following are additional considerations when benchmarking on OS/2:
- If paging occurs during the scenario, ensure that SWAPPER.DAT has returned to the original size.
- Re-boot the system for repeatability, if necessary.

Output from the benchmark program should include an identifier for each test, the iteration of the program execution, the statement number, and the timing for the execution. A summary of benchmarking results after a series of measurements might look like the following:

```
Test     Iter.   Stmt    Timing         SQL Statement
Numbr    Numbr   Numbr   (hh:mm:ss.ss)
 002      05      01     00:00:01.34    CONNECT TO SAMPLE
 002      05      10     00:02:08.15    OPEN cursor_01
 002      05      15     00:00:00.24    FETCH cursor_01
 002      05      15     00:00:00.23    FETCH cursor_01
 002      05      15     00:00:00.28    FETCH cursor_01
 002      05      15     00:00:00.21    FETCH cursor_01
 002      05      15     00:00:00.20    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      20     00:00:00.84    CLOSE cursor_01
 002      05      99     00:00:00.03    CONNECT RESET
```

Figure 36. Benchmark Sample Results

Note: The data in the above report is shown for illustration purposes only. It does **not** represent measured results.

Examining this report would indicate that the CONNECT (statement 01) took 1.34 seconds, the OPEN CURSOR (statement 10) took 2 minutes and 8.15 seconds, the FETCHES (statement 15) returned seven rows with the longest delay being .28 seconds, the CLOSE CURSOR (statement 20) took .84 seconds, and the CONNECT RESET (statement 99) took .03 seconds.

It might be beneficial for your program to output your data in a delimited ASCII format so that it could later be imported into a database table or a spreadsheet for further statistical analysis.

Sample output for a benchmark report might be:

```
     PARAMETER         VALUES FOR EACH BENCHMARK TEST
     TEST NUMBER       001     002     003    004     005
     locklist          63      63      63      63      63
 >>  buffpage          1000    1175    1250    1325    1400     <<
     maxappls          8       8       8       8       8
     applheapsz        48      48      48      48      48
     dbheap            128     128     128     128     128
     sortheap          256     256     256     256     256
     maxlocks          22      22      22      22      22
     stmtheap          1024    1024    1024    1024    1024
     SQL STMT          AVERAGE TIMINGS (seconds)
       01              01.34   01.34   01.35   01.35   01.36
       10              02.15   02.00   01.55   01.24   01.00
       15              00.22   00.22   00.22   00.22   00.22
       20              00.84   00.84   00.84   00.84   00.84
       99              00.03   00.03   00.03   00.03   00.03
```

Figure 37. Benchmark Sample Timings Report

Note: The data in the above report is shown for illustration purposes only. It does **not** represent any measured results.

Examining the data in this example shows that changing the *buffpage* parameter successively lowered the OPEN CURSOR times from 2.15 seconds to 1.00 second. (The assumption is that there is only one (1) buffer pool with the size (NPAGES) set to -1. This means the size of the buffer pool is controlled by the *buffpage* parameter.)

In summary, the following steps/iterations may be followed to benchmark a database application:

Step 1 Leave the database and database manager tuning parameters at their **default** values except for:
- Those parameters significant to the workload and the objectives of the test. (You rarely have enough time to perform benchmark

testing to tune all of the parameters, so you may want to start by using your best guess for some of the parameters and tune from that point.)
- Log sizes, which should be determined during unit and system testing of your application. (See "Size of Log Files (logfilsiz)" on page 389 for more information.)
- Any parameters that must be changed to enable your application to run (that is, the changes needed to prevent negative SQL return codes from such events as running out of memory for the statement heap).

Run your set of iterations for this initial case and calculate the average timing or CPU.

Step 2 Select one and only one tuning parameter to be tested, and change its value.

Step 3 Run another set of iterations and calculate the average timing or CPU.

Step 4 Depending on the results of the benchmark test, do one of the following:
- If performance improves, change the value of the same parameter and return to Step 3. Keep changing this parameter until the maximum benefit is shown.
- If performance degrades or remains unchanged, return the parameter to its previous value, return to Step 2, and select a new parameter. Repeat this procedure until all parameters have been tested.

  Note: If you were to graph the performance results, you would be looking for the point where the curve begins to plateau or decline.

You can write a driver program to help you with your benchmark testing. This driver program could be written using a language such as REXX or, for UNIX-based platforms, using shell scripts.

This driver program would execute the benchmark program, pass it the appropriate parameters, drive the test through multiple iterations, restore the environment to a consistent state, set up the next test with new parameter values, and collect/consolidate the test results. These driver programs can be flexible enough that they could be used to run the entire set of benchmark tests, analyze the results, and provide a report of the final and best parameter values for the given test.

Chapter 13. Configuring DB2

Configuration parameters are values that affect the operating characteristics of a database or database management system.

Database manager configuration parameters exist on servers and clients; however, only certain database manager configuration parameters can be set on the client. These parameters are a subset of the database management configuration parameters that can be set on the server. There are specific issues relating to configuration parameters depending on the type of DB2 Universal Database product you are using. For example, in DB2 Extended Enterprise Edition, one database manager configuration file is shared between all database partition servers in the instance.

Database configuration parameters only reside on a server.

DB2 has been designed with an extensive array of tuning and configuration parameters. These parameters fall into two general categories:
- "Database Manager Parameters" on page 313
- "Database Parameters" on page 319.

In addition to descriptions of the individual parameters, the following topics are available which are significantly affected by configuration parameters:
- "Tuning Configuration Parameters" on page 312.
- "Parameter Details by Function" on page 325 (each functional area has its own list of configuration parameters).
- "Appendix A. DB2 Registry and Environment Variables" on page 469.

  There may be performance-related environment or registry variables for your specific platform that you should consider using in addition to the performance-related configuration parameters.
- "Chapter 8. Operational Performance" on page 227.
- "Chapter 12. Benchmark Testing" on page 299.

You should review all of the parameter summaries in Table 17 on page 315 and Table 19 on page 321, and then focus on the descriptions and tuning of those which will provide you with the greatest benefit in your working environment.

Tuning Configuration Parameters

The disk space and memory allocated by the database manager on the basis of default values of the parameters may be sufficient to meet your needs. In some situations, however, you may not be able to achieve maximum performance using these default values.

Since the default values are oriented towards machines with relatively small memory and dedicated as database servers, you may need to modify them if your environment has:

- Large databases
- Large numbers of connections
- High performance requirements for a specific application
- Unique query or transaction loads or types
- Different machine configuration or usage.

Each transaction processing environment is unique in one or more aspects. These differences can have a profound impact on the performance of the database manager when using the default configuration. For this reason, you are strongly advised to tune your configuration for your environment.

Different types of applications and users have different response time requirements and expectations. Applications could range from simple data entry screens to strategic applications involving dozens of complex SQL statements accessing dozens of tables per unit of work. For example, response time requirements could vary considerably in a telephone customer service application versus a batch report generation application.

The other related topics can be used to help you benchmark your application to tune the configuration parameters:

- "Database Manager Parameters" on page 313
- "Database Parameters" on page 319
- "Parameter Details by Function" on page 325 (each functional area has its own list of configuration parameters)
- "Chapter 8. Operational Performance" on page 227
- "Chapter 12. Benchmark Testing" on page 299
- Database system monitor element descriptions in the *System Monitor Guide and Reference*.

Database Manager Parameters

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

In a 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.

The `db2systm` file cannot be directly edited. It can only be changed or viewed using a supplied API or by a tool which calls that API.

Attention: If you edit the file using a method other than those provided by the product, you may make your system unusable. **We strongly recommend** that you do not change this file using methods other than those documented and supported by DB2.

You may use one of the following methods to reset, update, and view the database manager configuration parameters:

- Using the DB2 Control Center. The DB2 Control Center provides the Configure Instance notebook, which you can use to set the database manager configuration parameters on either a client or a server. The DB2 Control Center also provides the Performance Configuration wizard to alter the value of configuration parameters on a server. This wizard generates values to parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database. See the online help available with the Control Center for information on using these interfaces.
- Using the command line processor. Commands to change the settings can be quickly and conveniently entered. Refer to the *Command Reference* for more information about the following commands:
  - GET DATABASE MANAGER CONFIGURATION (or GET DBM CFG)

  – UPDATE DATABASE MANAGER CONFIGURATION (or UPDATE DBM CFG)
  – RESET DATABASE MANAGER CONFIGURATION (or RESET DBM CFG).
- Using the application programming interfaces (APIs). The APIs can easily be called from an application. Refer to the *Administrative API Reference* for more information.
- Using the Client Configuration Assistant. You can only use the Client Configuration Assistant to set the database manager configuration parameters on a client.

After changing the parameters, the database manager must be stopped (`db2stop`) and then restarted (`db2start`) for the new parameter values to take effect. For clients, changes in the database manager configuration parameters take effect the next time the client connects to a server. While new parameter values are not immediately effective, viewing the parameter settings will always show the latest updates.

Note: You do not need to restart the database manager if you update the value of the *dft_monswitches* parameter; this parameter is updated automatically when you change its value.

Database Manager Configuration Parameter Summary

The following table lists the parameters in the database manager configuration file for database servers. When changing the database manager configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

The column "Performance Impact" in the following table provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High —** indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters; which, in some cases, will mean that you accept the default provided.
- **Medium —** indicates the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low —** indicates that the parameter has a less general or less significant impact on performance.

- **None —** indicates that the parameter does not directly impact performance. While you do not have to tune these parameters for performance, they can be very important for other aspects of your system configuration, such as enabling communication support.

Table 17. Configurable Database Manager Configuration Parameters

Parameter	**Performance Impact**	**Additional Information**
agentpri	High	"Priority of Agents (agentpri)" on page 378
agent_stack_sz	Low	"Agent Stack Size (agent_stack_sz)" on page 349
aslheapsz	High	"Application Support Layer Heap Size (aslheapsz)" on page 353
audit_buf_sz	High	"Audit Buffer Size (audit_buf_sz)" on page 361
authentication	Low	"Authentication Type (authentication)" on page 461
backbufsz	Medium	"Default Backup Buffer Size (backbufsz)" on page 334
catalog_noauth	None	"Cataloging Allowed without Authority (catalog_noauth)" on page 462
comm_bandwidth	Medium	"Communications Bandwidth (comm_bandwidth)" on page 450
conn_elapse	Medium	"Connection Elapse Time (conn_elapse)" on page 438
cpuspeed	Low (see note)	"CPU Speed (cpuspeed)" on page 451
datalinks	Low	"Enable Data Links Support (datalinks)" on page 416
dft_account_str	None	"Default Charge-Back Account (dft_account_str)" on page 456
dft_client_adpt	None	"Default Client Adapter Number (dft_client_adpt)" on page 434
dft_client_comm	None	"Default Client Communication Protocol (dft_client_comm)" on page 433
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_uow*	Medium	"Default Database System Monitor Switches (dft_monswitches)" on page 448
dftdbpath	None	"Default Database Path (dftdbpath)" on page 463

Table 17. Configurable Database Manager Configuration Parameters (continued)

Parameter	**Performance Impact**	**Additional Information**
diaglevel	Low	"Diagnostic Error Capture Level (diaglevel)" on page 446
diagpath	None	"Diagnostic Data Directory Path (diagpath)" on page 446
dir_cache	Medium	"Directory Cache Support (dir_cache)" on page 359
dir_obj_name	None	"Object Name in DCE Namespace (dir_obj_name)" on page 431
dir_path_name	None	"Directory Path Name in DCE Namespace (dir_path_name)" on page 431
dir_type	None	"Directory Services Type (dir_type)" on page 430
discover	Medium	"Discovery Mode (discover)" on page 435
discover_comm	Low	"Search Discovery Communications Protocols (discover_comm)" on page 436
discover_inst	Low	"Discover Server Instance (discover_inst)" on page 437
dos_rqrioblk	High	"DOS Requester I/O Block Size (dos_rqrioblk)" on page 356
drda_heap_sz	Low	"DRDA Heap Size (drda_heap_sz)" on page 347
fcm_num_anchors	High	"Number of FCM Message Anchors (fcm_num_anchors)" on page 438
fcm_num_buffers	High	"Number of FCM Buffers (fcm_num_buffers)" on page 439
fcm_num_connect	High	"Number of FCM Connection Entries (fcm_num_connect)" on page 440
fcm_num_rqb	High	"Number of FCM Request Blocks (fcm_num_rqb)" on page 441
federated	Medium	"Federated Database System Support (federated)" on page 457
fileserver	None	"IPX/SPX File Server Name (fileserver)" on page 427
indexrec	Medium	"Index Re-creation Time (indexrec)" on page 401
initdari_jvm	Medium	"Initialize DARI Process with JVM (initdari_jvm)" on page 387
intra_parallel	High	"Enable Intra-Partition Parallelism (intra_parallel)" on page 445
ipx_socket	None	"IPX/SPX Socket Number (ipx_socket)" on page 429

Table 17. Configurable Database Manager Configuration Parameters (continued)

Parameter	Performance Impact	Additional Information
java_heap_sz	High	"Maximum Java Interpreter Heap Size (java_heap_sz)" on page 362
jdk11_path	None	"Java Development Kit 1.1 Installation Path (jdk11_path)" on page 456
keepdari	Medium	"Keep DARI Process Indicator (keepdari)" on page 385
maxagents	Medium	"Maximum Number of Agents (maxagents)" on page 379
maxcagents	Medium	"Maximum Number of Concurrent Agents (maxcagents)" on page 380
max_connretries	Medium	"Node Connection Retries (max_connretries)" on page 442
max_coordagents	Medium	"Maximum Number of Coordinating Agents (max_coordagents)" on page 381
maxdari	Medium	"Maximum Number of DARI Processes (maxdari)" on page 386
max_logicagents	Medium	"Maximum Number of Logical Agents (max_logicagents)" on page 382
max_querydegree	High	"Maximum Query Degree of Parallelism (max_querydegree)" on page 443
max_time_diff	Medium	"Maximum Time Difference Among Nodes (max_time_diff)" on page 442
maxtotfilop	Medium	"Maximum Total Files Open per Application (maxtotfilop)" on page 377
min_priv_mem	Medium	"Minimum Committed Private Memory (min_priv_mem)" on page 351
mon_heap_sz	Low	"Database System Monitor Heap Size (mon_heap_sz)" on page 358
nname	None	"NetBIOS Workstation Name (nname)" on page 425
notifylevel	Low	"Notify Level (notifylevel)" on page 447
numdb	Low	"Maximum Number of Concurrently Active Databases (numdb)" on page 451
num_initagents	Medium	"Initial Number of Agents in Pool (num_initagents)" on page 384
num_initdaris	Medium	"Initial Number of Fenced DARI Processes in Pool (num_initdaris)" on page 388
num_poolagents	High	"Agent Pool Size (num_poolagents)" on page 383

Table 17. Configurable Database Manager Configuration Parameters (continued)

Parameter	**Performance Impact**	**Additional Information**
objectname	None	"IPX/SPX DB2 Server Object Name (objectname)" on page 428
priv_mem_thresh	Medium	"Private Memory Threshold (priv_mem_thresh)" on page 351
query_heap_sz	Medium	"Query Heap Size (query_heap_sz)" on page 346
restbufsz	Medium	"Default Restore Buffer Size (restbufsz)" on page 334
resync_interval	None	"Transaction Resync Interval (resync_interval)" on page 407
route_obj_name	None	"Routing Information Object Name (route_obj_name)" on page 432
rqrioblk	High	"Client I/O Block Size (rqrioblk)" on page 355
sheapthres	High	"Sort Heap Threshold (sheapthres)" on page 342
spm_log_file_sz	Low	"Sync Point Manager Log File Size (spm_log_file_sz)" on page 409
spm_log_path	Medium	"Sync Point Manager Log File Path (spm_log_path)" on page 408
spm_max_resync	Low	"Sync Point Manager Resync Agent Limit (spm_max_resync)" on page 410
spm_name	None	"Sync Point Manager Name (spm_name)" on page 408
ss_logon	None	"LOGON Required for DB2START/DB2STOP (ss_logon)" on page 464
start_stop_time	Low	"Start and Stop Timeout (start_stop_time)" on page 443
svcename	None	"TCP/IP Service Name (svcename)" on page 426
sysadm_group	None	"System Administration Authority Group Name (sysadm_group)" on page 458
sysctrl_group	None	"System Control Authority Group Name (sysctrl_group)" on page 459
sysmaint_group	None	"System Maintenance Authority Group Name (sysmaint_group)" on page 460
tm_database	None	"Transaction Manager Database Name (tm_database)" on page 406
tp_mon_name	None	"Transaction Processor Monitor Name (tp_mon_name)" on page 453
tpname	None	"APPC Transaction Program Name (tpname)" on page 427

Table 17. Configurable Database Manager Configuration Parameters (continued)

Parameter	Performance Impact	Additional Information
trust_allclnts	None	"Trust All Clients (trust_allclnts)" on page 464
trust_clntauth	None	"Trusted Clients Authentication (trust_clntauth)" on page 466
udf_mem_sz	Low	"UDF Shared Memory Set Size (udf_mem_sz)" on page 348
Note: The *cpuspeed* parameter can have a significant impact on performance but you should use the default value, except in very specific circumstances, as documented in the parameter description.		

Table 18. Informational Database Manager Configuration Parameters

Parameter	Additional Information
nodetype	"Machine Node Type (nodetype)" on page 455
release	"Configuration File Release Level (release)" on page 411

Database Parameters

Parameters for an individual database are stored in a configuration file named `SQLDBCON`. This file is stored along with other control files for the database in the `SQLnnnnn` directory, where `nnnnn` is a number assigned when the database was created. (For more information about the location of this directory, refer to "Database Physical Directories" in the *Administration Guide: Planning*.) 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.

The `SQLDBCON` file cannot be directly edited, and can only be changed or viewed via a supplied API or by a tool which calls that API.

Attention: If you edit the file using a method other than those provided by DB2, you may make the database unusable. **We strongly recommend** that you do not change this file using methods other than those documented and supported by DB2.

You may use one of the following three methods to reset, update, and view the database configuration parameters:

- Using the Control Center. The DB2 Control Center provides both the Configure Database notebook and the Performance Configuration wizard to alter the value of configuration parameters. This wizard generates values to parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database. See the online help available with the Control Center for information on using these interfaces.

In a partitioned database environment, the `SQLDBCON` file exists for each database partition. The Control Center Configure Database notebook will change the value on all partitions if you launch the notebook from the database object in the tree view of the Control Center. If you launch the notebook from a database partition object, then it will only change the values for that partition.

Note: The Performance Configuration wizard is not available in the partitioned database environment.

- Using the command line processor. Commands to change the settings can be quickly and conveniently entered. Refer to the *Command Reference* for more information about the following commands:
  - GET DATABASE CONFIGURATION (or GET DB CFG)
  - UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (or RESET DB CFG)
- Using the application programming interfaces (APIs). The APIs can easily be called from a host-language program. Refer to the *Administrative API Reference* for more information.

Updates to most changeable parameters will not take effect while applications are connected to the database. All applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) Then, at the first new connect to the database, the changes will take effect. You should note that some parameter changes, such as *newlogpath*, *logfilsiz* and *logprimary*, may take a noticeable amount of time to take effect due to the overhead associated with allocating space. You may wish to make a test connection to the database so the change will be made at the time of the test connection and any overhead will not affect other users. If you are concerned about the overhead as discussed here, consider using the ACTIVATE DATABASE command as described in the *Command Reference*.

Note: You do not need to disconnect from the database if you update the value of the *mincommit* parameter; this parameter is updated automatically when you change its value.

Changing some database configuration parameters can influence the access plan chosen by the SQL optimizer. These database parameters are discussed in "Configuration Parameters Affecting Query Optimization" on page 87. After changing any of the parameters discussed there, you should consider rebinding your applications to ensure the best access plan is being used for your SQL statements.

While new parameter values may not be immediately effective, viewing the parameter settings will always show the latest updates.

Note: A number of database configuration parameters (for example, *userexit*) are described as having acceptable values of either "Yes" or "No", or "On" or "Off" in the help and other DB2 books. To clarify what may be confusing, "Yes" should be considered equivalent to "On" and "No" should be considered equivalent to "Off".

Database Configuration Parameter Summary

The following table lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

The column "Performance Impact" in the following table provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High —** indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters; which, in some cases, will mean that you accept the default provided.
- **Medium —** indicates the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low —** indicates that the parameter has a less general or less significant impact on performance.
- **None —** indicates that the parameter does not directly impact performance. While you do not have to tune these parameters for performance, they can be very important for other aspects of your system configuration, such as enabling communication support.

Table 19. Configurable Database Configuration Parameters

Parameter	Performance Impact	Additional Information
app_ctl_heap_sz	Medium	"Application Control Heap Size (app_ctl_heap_sz)" on page 340
applheapsz	Medium	"Application Heap Size (applheapsz)" on page 345
audit_buf_sz	Medium	"Audit Buffer Size (audit_buf_sz)" on page 361
autorestart	Low	"Auto Restart Enable (autorestart)" on page 401

Table 19. Configurable Database Configuration Parameters (continued)

Parameter	Performance Impact	Additional Information
avg_appls	High	"Average Number of Active Applications (avg_appls)" on page 375
buffpage	High (when active)	"Buffer Pool Size (buffpage)" on page 327
catalogcache_sz	Medium	"Catalog Cache Size (catalogcache_sz)" on page 331
chngpgs_thresh	High	"Changed Pages Threshold (chngpgs_thresh)" on page 366
copyprotect	None	"Copy Protection Enable (copyprotect)" on page 414
dbheap	Medium	"Database Heap (dbheap)" on page 329
dft_degree	High	"Default Degree (dft_degree)" on page 421
dft_extent_sz	Medium	"Default Extent Size of Table Spaces (dft_extent_sz)" on page 372
dft_loadrec_ses	Medium	"Default Number of Load Recovery Sessions (dft_loadrec_ses)" on page 402
dft_prefetch_sz	Medium	"Default Prefetch Size (dft_prefetch_sz)" on page 370
dft_queryopt	Medium	"Default Query Optimization Class (dft_queryopt)" on page 421
dft_refresh_age	Medium	"Default Refresh Age (dft_refresh_age)" on page 422
dft_sqlmathwarn	None	"Continue upon Arithmetic Exceptions (dft_sqlmathwarn)" on page 419
dir_obj_name	None	"Object Name in DCE Namespace (dir_obj_name)" on page 431
discover_db	Medium	"Discover Database (discover_db)" on page 434
dlchktime	Medium	"Time Interval for Checking Deadlock (dlchktime)" on page 363
dl_expint	None	"Data Links Access Token Expiry Interval (dl_expint)" on page 414
dl_num_copies	None	"Data Links Number of Copies (dl_num_copies)" on page 415
dl_time_drop	None	"Data Links Time After Drop (dl_time_drop)" on page 415
dl_token	Low	"Data Links Token Algorithm (dl_token)" on page 416
dl_upper	None	"Data Links Token in Upper Case (dl_upper)" on page 416

Table 19. Configurable Database Configuration Parameters (continued)

Parameter	Performance Impact	Additional Information
dyn_query_mgmt	Low	“Dynamic SQL Query Management (dyn_query_mgmt)” on page 411
estore_seg_sz	Medium	“Extended Storage Memory Segment Size (estore_seg_sz)” on page 372
indexrec	Medium	“Index Re-creation Time (indexrec)” on page 401
indexsort	Low (see 324)	“Index Sort Flag (indexsort)” on page 369
locklist	High when it affects escalation	“Maximum Storage for Lock List (locklist)” on page 335
locktimeout	Medium	“Lock Timeout (locktimeout)” on page 365
logbufsz	High	“Log Buffer Size (logbufsz)” on page 332
logfilsiz	Medium	“Size of Log Files (logfilsiz)” on page 389
logprimary	Medium	“Number of Primary Log Files (logprimary)” on page 391
logretain	Low	“Log Retain Enable (logretain)” on page 399
logsecond	Medium	“Number of Secondary Log Files (logsecond)” on page 392
maxappls	Medium	“Maximum Number of Active Applications (maxappls)” on page 374
maxfilop	Medium	“Maximum Database Files Open per Application (maxfilop)” on page 376
maxlocks	High when it affects escalation	“Maximum Percent of Lock List Before Escalation (maxlocks)” on page 364
mincommit	High	“Number of Commits to Group (mincommit)” on page 395
newlogpath	Low	“Change the Database Log Path (newlogpath)” on page 393
num_db_backups	None	“Number of Database Backups (num_db_backups)” on page 403
num_estore_segs	Medium	“Number of Extended Storage Memory Segments (num_estore_segs)” on page 373
num_freqvalues	Low	“Number of Frequent Values Retained (num_freqvalues)” on page 422
num_iocleaners	High	“Number of Asynchronous Page Cleaners (num_iocleaners)” on page 367
num_ioservers	High	“Number of I/O Servers (num_ioservers)” on page 369

Table 19. Configurable Database Configuration Parameters (continued)

Parameter	**Performance Impact**	**Additional Information**
num_quantiles	Low	"Number of Quantiles for Columns (num_quantiles)" on page 423
pckcachesz	High	"Package Cache Size (pckcachesz)" on page 338
rec_his_retentn	None	"Recovery History Retention Period (rec_his_retentn)" on page 404
seqdetect	High	"Sequential Detection Flag (seqdetect)" on page 370
softmax	Medium	"Recovery Range and Soft Checkpoint Interval (softmax)" on page 397
sortheap	High	"Sort Heap Size (sortheap)" on page 342
stat_heap_sz	Low	"Statistics Heap Size (stat_heap_sz)" on page 345
stmtheap	Medium	"Statement Heap Size (stmtheap)" on page 344
tsm_mgmtclass	None	"Tivoli Storage Manager Management Class (tsm_mgmtclass)" on page 404
tsm_nodename	None	"Tivoli Storage Manager Node Name (tsm_nodename)" on page 405
tsm_owner	None	"Tivoli Storage Manager Owner Name (tsm_owner)" on page 406
tsm_password	None	"Tivoli Storage Manager Password (tsm_password)" on page 405
userexit	Low	"User Exit Enable (userexit)" on page 399
util_heap_sz	Low	"Utility Heap Size (util_heap_sz)" on page 333
Note: Changing the *indexsort* parameter to a value other than the default can have a negative impact on the performance of creating indexes. You should always try to use the default for this parameter.		

Table 20. Informational Database Configuration Parameters

Parameter	**Additional Information**
backup_pending	"Backup Pending Indicator (backup_pending)" on page 417
codepage	"Code Page for the Database (codepage)" on page 413
codeset	"Codeset for the Database (codeset)" on page 413
collate_info	"Collating Information (collate_info)" on page 413
country	"Country code for the Database (country)" on page 412

Table 20. Informational Database Configuration Parameters (continued)

Parameter	Additional Information
database_consistent	"Database is Consistent (database_consistent)" on page 417
database_level	"Database Release Level (database_level)" on page 412
log_retain_status	"Log Retain Status Indicator (log_retain_status)" on page 418
loghead	"First Active Log File (loghead)" on page 395
logpath	"Location of Log Files (logpath)" on page 395
multipage_alloc	"MultiPage File Allocation Enabled (multipage_alloc)" on page 419
numsegs	"Default Number of SMS Containers (numsegs)" on page 371
release	"Configuration File Release Level (release)" on page 411
restore_pending	"Restore Pending (restore_pending)" on page 418
rollfwd_pending	"Roll Forward Pending Indicator (rollfwd_pending)" on page 418
territory	"Territory for the Database (territory)" on page 412
user_exit_status	"User Exit Status Indicator (user_exit_status)" on page 418

Parameter Details by Function

This following sections provide additional details to assist in understanding and tuning the different configuration parameters. This discussion of the individual parameters is organized based on their function or purpose:

- "Capacity Management" on page 326
- "Logging and Recovery" on page 389
- "Database Management" on page 410
- "Communications" on page 425
- "Parallel" on page 437
- "Instance Management" on page 445.

The discussion of each parameter includes the following information:

Configuration Type Indicates which configuration file contains the setting for the parameter:

- Database manager (which affects an instance of the database manager and all databases defined within that instance)
- Database (which affects a specific database)

Parameter Type

Indicates whether or not you can change the parameter value:

- *Configurable*

  A range of values are possible and the parameter may need to be tuned based on the database administrator's knowledge of the applications and/or from benchmarking experience.
- *Informational*

  These parameters are changed only by the database manager itself and will contain information such as the release of DB2 that a database was created under or an indication that a required backup is pending.

Capacity Management

There are a number of configuration parameters at both the database and database manager levels that can impact the throughput on your system. These parameters are categorized in the following groups:

- "Database Shared Memory"
- "Application Shared Memory" on page 340
- "Agent Private Memory" on page 341
- "Agent/Application Communication Memory" on page 353
- "Database Manager Instance Memory" on page 358
- "Locks" on page 362
- "I/O and Storage" on page 366
- "Agents" on page 373
- "Database Application Remote Interface (DARI)" on page 385.

For an introduction to DB2's memory management, see "How DB2 Uses Memory" on page 227.

Database Shared Memory

The following parameters affect the database global memory allocated on your system:

- "Buffer Pool Size (buffpage)" on page 327.

- “Database Heap (dbheap)” on page 329.
- “Catalog Cache Size (catalogcache_sz)” on page 331.
- “Log Buffer Size (logbufsz)” on page 332.
- “Utility Heap Size (util_heap_sz)” on page 333.
- “Default Backup Buffer Size (backbufsz)” on page 334.
- “Default Restore Buffer Size (restbufsz)” on page 334.
- “Maximum Storage for Lock List (locklist)” on page 335.
- “Package Cache Size (pckcachesz)” on page 338.

See “How DB2 Uses Memory” on page 227 for information about how database global memory relates to the rest of the memory allocated by the database manager.

Buffer Pool Size (buffpage)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	**UNIX 32-bit platforms** 1 000 [2 — 524 288] **UNIX 64-bit platforms** 1 000 [2 — 2 147 483 647] **OS/2 and Windows NT** 250 [2 — 524 288]
Unit of Measure	Pages
When Allocated	When the first application connects to the database
When Freed	When last application disconnects from the database
Related Parameters	• “Changed Pages Threshold (chngpgs_thresh)” on page 366 • “Database Heap (dbheap)” on page 329 • “Number of Asynchronous Page Cleaners (num_iocleaners)” on page 367

Each database has at least one buffer pool (IBMDEFAULTBP, which is created when the database is created), and can have more. All buffer pools reside in global memory, which is available to all applications using the database. The memory is allocated on the machine where the database is located. If the buffer pools are large enough to keep the required data in memory, less disk activity will occur. Conversely, if the buffer pools are not large enough, the overall performance of the database can be severely curtailed and the database manager can become I/O-bound as a result of a high amount of disk activity (I/O) required to process the data your application requires.

The *buffpage* parameter controls the size of a buffer pool when the CREATE BUFFERPOOL or ALTER BUFFERPOOL statement was run with NPAGES -1; otherwise, the *buffpage* parameter is ignored and the buffer pool will be created with the number of pages specified by the NPAGES parameter.

To determine whether the *buffpage* parameter is active for a buffer pool, do a:

```
SELECT * from SYSCAT.BUFFERPOOLS.
```

Each buffer pool that has an NPAGES value of -1 uses *buffpage*.

There is a trade-off between the buffer pool size and the memory allocations of other system users. Memory requirements of database servers are so important on multi-user high transaction rate servers, that database servers and file or communication servers are often separated and reside on different machines.

If your queries access nicknames, consider increasing the buffer pool size when:
- The optimizer decides that most or all operations are completed locally. When a query is processed, the optimizer will usually push down operations to the data source where possible. As an example, a GROUP BY operator is usually evaluated at the data source. It is possible, however, that materializing the table at DB2 and performing an operation locally is the least cost route. This situation could occur if the DB2 server workstation is more powerful than the data source workstation.
- Sort operations must be completed locally. Queries containing nicknames are sorted according to the DB2 collating sequence. If a data source does not have the same collating sequence, all sort operations are performed locally.

All buffer pools are allocated when the first application connects to the database, or when the database is explicitly activated. As an application requests data out of the database, pages containing that data are transferred to one of the buffer pools from disk. (Note that database data is stored in pages within the tables on the disk.) Pages are not written back to disk until the page is changed and one of the following occurs:
- All applications disconnect from the database
- The database is explicitly deactivated
- The database quiesces (that is, all connected applications have committed)

- Its space is required for another page that needs to be read into the buffer pool
- A page cleaner is available (*num_iocleaners*) and is activated by the database manager.

Recommendations:

- Instead of using the *buffpage* configuration parameter, you can use the CREATE BUFFERPOOL and ALTER BUFFERPOOL SQL statements to create and change buffer pools and their sizes.
- The size of the buffer pool is used by the optimizer in determining access plans. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.
- Because the sizes of all the buffer pools can have a major impact on performance, you should consider the following factors to ensure that excessive page swapping does not occur:
  - The amount of installed memory on your machine.
  - The memory required by other applications running concurrently with the database manager on the same machine.

  Page swapping results when there is not enough memory to hold the page that is being accessed. The result is that the page is written (“swapped”) to temporary disk storage to make room for the other page. When the page on the temporary disk storage is needed, it is “swapped back” into memory.
- You may wish to allocate as much as 75% of the machine’s memory to the database buffer pools when you have the following:
  - Multiple users
  - A machine used only as a database server
  - A large amount of repeated access to the same data and index pages
  - One database on the machine.
- For every buffer pool page allocated, some space is used in the database heap for internal control structures.

  If the total size of the buffer pool (or buffer pools) is increased, you may also need to increase *dbheap*.
- If the data source collating sequence matches the DB2 collating sequence, ensure that the server option collating_sequence is set to indicate so.

You may use the database system monitor to calculate the buffer pool hit ratio, which can help you tune your buffer pools. Refer to the *System Monitor Guide and Reference*.

Database Heap (dbheap)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	

	UNIX	1200 [32 – 524 288]
	OS/2 and Windows NT Database server with local and remote clients	600 [32 – 524 288]
	OS/2 and Windows NT Database server with local clients	300 [32 – 524 288]
Unit of Measure	Pages (4 KB)	
When Allocated	First connection to the database	
When Freed	When last application disconnects from the database	
Related Parameters	• "Catalog Cache Size (catalogcache_sz)" on page 331 • "Log Buffer Size (logbufsz)" on page 332	

There is one database heap per database, and the database manager uses it on behalf of all applications connected to the database. It contains control block information for tables, indexes, table spaces, and buffer pools. It also contains space for the log buffer (*logbufsz*), and the catalog cache (*catalogcache_sz*). Therefore, the size of the heap will be dependent on the number of control blocks stored in the heap at a given time. The control block information is kept in the heap until all applications disconnect from the database.

The minimum amount the database manager needs to get started is allocated at the first connection. The data area is expanded as needed up to the maximum specified by *dbheap*.

Recommendation: This value will need to be increased when an application receives an error indicating that there is not enough storage available in the database heap to process the statement.

You may use the database system monitor to track the highest amount of memory that was used for the database heap. See the *db_heap_top (maximum database heap allocated)* monitor element description in the *System Monitor Guide and Reference* for more information.

When setting this parameter, you should consider:
- The value of *logbufsz*, because the log buffer is allocated from the database heap.
- The value of *catalogcache_sz*, because the catalog cache is allocated from the database heap.

Catalog Cache Size (catalogcache_sz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	
	UNIX 64 [1 – 60 000]
	OS/2 and Windows NT Database server with local and remote clients 32 [1 – 60 000]
	OS/2 and Windows NT Database server with local clients 16 [1 – 60 000]
Unit of Measure	Pages (4 KB)
Related Parameters	• "Database Heap (dbheap)" on page 329 • "Log Buffer Size (logbufsz)" on page 332 • "Application Control Heap Size (app_ctl_heap_sz)" on page 340

This parameter indicates the maximum amount of space that the catalog cache can use from the database heap (*dbheap*). The catalog cache is used to store table descriptor information that is used when a table, view or alias is referenced during the compilation of an SQL statement.

Use of this cache can help improve performance of binding SQL statements (including dynamic SQL), if the same tables, views, or aliases have been referenced in previous statements. Descriptor information for declared temporary tables is not stored in the catalog cache; instead the application control heap is used.

Running any DDL statements against a table will purge that table's entry in the catalog cache. Otherwise a table entry is kept in the cache until space is needed for a different table, but it will not be removed from the cache until any units of work referencing that table have completed.

Recommendation: Start with the default value and tune it by using the database system monitor.

See the *System Monitor Guide and Reference* for information about the following monitor elements:
- *cat_cache_lookups* (catalog cache lookups)
- *cat_cache_inserts* (catalog cache inserts)
- *cat_cache_overflows* (catalog cache overflows)
- *cat_cache_heap_full* (catalog cache heap full)

These database system monitor elements can help you determine whether you should adjust this configuration parameter. When tuning this parameter, you should increase it in small increments, for example, two pages at a time.

Note: The catalog cache only exists at the catalog node in a multinode environment.

In general, more cache space is required if a unit of work contains several dynamic SQL statements or if you are binding packages that contain a lot of static SQL statements.

When you set the size of the catalog cache, also consider the size of the log files (*logbufsz*), because both *catalogcache_sz* and *logbufsz* are allocated from the database heap (*dbheap*).

Log Buffer Size (logbufsz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	**UNIX 32-bit platforms** 8 [4 — 4 096] **UNIX 64-bit platforms** 8 [4 — 65 535] **OS/2 and Windows NT** 8 [4 — 4 096]
Unit of Measure	Pages (4 KB)
Related Parameters	• “Catalog Cache Size (catalogcache_sz)” on page 331 • “Database Heap (dbheap)” on page 329 • “Number of Commits to Group (mincommit)” on page 395

This parameter allows you to specify the amount of the database heap (defined by the *dbheap* parameter) to use as a buffer for log records before writing these records to disk. The log records are written to disk when one of the following occurs:
- A transaction commits or a group of transactions commit, as defined by the *mincommit* configuration parameter
- The log buffer is full
- As a result of some other internal database manager event.

This parameter must also be less than or equal to the *dbheap* parameter. Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently and more log records will be written at each time.

Recommendation: Increase the size of this buffer area if there is considerable read activity on a dedicated log disk, or there is high disk utilization. When increasing the value of this parameter, you should also consider the *dbheap* parameter since the log buffer area uses space controlled by the *dbheap* parameter.

You may use the database system monitor to determine how much of the log buffer space is used for a particular transaction (or unit of work).

For more information refer to the *log_space_used* (unit of work log space used) monitor element description in the *System Monitor Guide and Reference*.

When you set the log buffer size, also consider the size of the catalog cache (*catalogcache_sz*), because both *logbufsz_sz* and *catalogcache_sz* are allocated from the database heap (*dbheap*).

Utility Heap Size (util_heap_sz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	5000 [16 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	As required by the database manager utilities
When Freed	When the utility no longer needs the memory
Related Parameters	• "Default Backup Buffer Size (backbufsz)" on page 334 • "Default Restore Buffer Size (restbufsz)" on page 334

This parameter indicates the maximum amount of memory that can be used simultaneously by the BACKUP, RESTORE and LOAD and load recovery utilities.

Recommendation: Use the default value unless your utilities run out of space, in which case you should increase this value. If memory on your system is constrained, you may wish to lower the value of this parameter to limit the memory used by the database utilities. If the parameter is set too low, you may not be able to concurrently run utilities. You need to set this parameter large enough to accommodate all of the buffers that you want to allocate for the concurrent utilities.

Default Backup Buffer Size (backbufsz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	1024 [8 — 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the backup utility is called
When Freed	When the backup utility completes its processing
Related Parameters	• "Default Restore Buffer Size (restbufsz)" • "Utility Heap Size (util_heap_sz)" on page 333

This parameter specifies the size of the buffer used when backing up the database if the buffer size is not explicitly specified when calling the backup utility. For more information about the backup utility, refer to the *Command Reference*.

When backing up a database, the data is first copied to an internal buffer. Data is then written from this buffer to the backup media when the buffer is full.

Tuning this buffer size can help improve the performance of the backup utility as well as minimize the impact on the performance of other concurrent database operations.

Default Restore Buffer Size (restbufsz)

Configuration Type	Database manager
Applies to	

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type	Configurable
Default [Range]	1024 [16 — 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the restore utility is called
When Freed	When the restore utility completes its processing

Related Parameters

- “Default Backup Buffer Size (backbufsz)” on page 334
- “Utility Heap Size (util_heap_sz)” on page 333

This parameter specifies the size of the buffer used when restoring the database if a buffer size is not explicitly specified when calling the restore utility. For more information about the restore utility, refer to the *Command Reference*.

When restoring a database, the data is first copied from the backup media to an internal buffer. Data is then written from this buffer to the target database media when the buffer is full.

Tuning this buffer size can help improve the performance of the restore database utility as well as minimize the impact on the performance of other concurrent database operations.

Maximum Storage for Lock List (locklist)

Configuration Type	Database
Parameter Type	Configurable

Default [Range]

UNIX	100 [4 – 60 000]
OS/2 and NT Database server with local and remote clients	50 [4 – 60 000]
OS/2 and NT Database server with local clients	25 [4 – 60 000]

Unit of Measure	Pages (4 KB)
When Allocated	When the first application connects to the database
When Freed	When last application disconnects from the database
Related Parameters	• "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 364 • "Maximum Number of Active Applications (maxappls)" on page 374

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database and it contains the locks held by all applications concurrently connected to the database. Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked. The database manager may also acquire locks for internal use.

Each lock requires 36 or 72 bytes of the lock list, depending on whether other locks are held on the object:
- 72 bytes are required to hold a lock on an object that has no other locks held on it
- 36 bytes are required to record a lock on an object that has an existing lock held on it.

When the percentage of the lock list used by one application reaches *maxlocks*, the database manager will perform lock escalation, from row to table, for the locks held by the application (described below). Although the escalation process itself does not take much time, locking entire tables (versus individual rows) decreases concurrency, and overall database performance may decrease for subsequent accesses against the affected tables. Suggestions of how to control the size of the lock list are:
- Perform frequent COMMITs to release locks.
- When performing many updates, lock the entire table before updating (using the SQL LOCK TABLE statement). This will use only one lock, keeps others from interfering with the updates, but does reduce concurrency of the data.

  You can also use the LOCKSIZE parameter of the ALTER TABLE statement to control how locking is done for a specific table. For details, refer to the *SQL Reference*.

  Use of the Repeatable Read isolation level may result in an automatic table lock. For more information on isolation levels, see "Chapter 3. Application Considerations" on page 41.

- Use the Cursor Stability isolation level when possible to decrease the number of share locks held. If application integrity requirements are not compromised use Uncommitted Read instead of Cursor Stability to further decrease the amount of locking.

Once the lock list is full, performance can degrade since lock escalation will generate more table locks and fewer row locks, thus reducing concurrency on shared objects in the database. Additionally there may be more deadlocks between applications (since they are all waiting on a limited number of table locks), which will result in transactions being rolled back. Your application will receive an SQLCODE of -912 when the maximum number of lock requests has been reached for the database.

Recommendation: If lock escalations are causing performance concerns you may need to increase the value of this parameter or the *maxlocks* parameter. You may use the database system monitor to determine if lock escalations are occurring.

For more information see the *lock_escals (lock escalations)* monitor element description in the *System Monitor Guide and Reference*.

The following steps may help in determining the number of pages required for your lock list:

1. Calculate a lower bound for the size of your lock list:
   ```
   (512 * 36 * maxappls) / 4096
   ```
   where 512 is an estimate of the average number of locks per application and 36 is the number of bytes required for each lock against an object that has an existing lock.
2. Calculate an upper bound for the size of your lock list:
   ```
   (512 * 72 * maxappls) / 4096
   ```
   where 72 is the number of bytes required for the first lock against an object.
3. Estimate the amount of concurrency you will have against your data and based on your expectations, choose an initial value for *locklist* that falls between the upper and lower bounds that you have calculated.
4. Using the database system monitor, as described below, tune the value of this parameter.

You may use the database system monitor to determine the maximum number of locks held by a given transaction.

For more information see the *locks_held_top (maximum number of locks held)* monitor element description in the *System Monitor Guide and Reference*.

This information can help you validate or adjust the estimated number of locks per application. In order to perform this validation, you will have to sample several applications, noting that the monitor information is provided at a transaction level, not an application level.

You may also want to increase *locklist* if *maxappls* is increased, or if the applications being run perform infrequent commits.

You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

For more information on application performance and influencing query optimization, see "Part 2. Tuning Application Performance" on page 39.

Package Cache Size (pckcachesz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	
	UNIX 32-bit platforms -1 [-1, 32 — 64 000]
	UNIX 64-bit platforms -1 [-1, 32 — 524 288]
	OS/2 and Windows NT -1 [-1, 32 — 64 000]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is initialized
When Freed	When the database is shutdown

This parameter is allocated out of the database global memory, and is used for caching static and dynamic SQL statements on a database. In a partitioned databasesystem, there is one package cache for each database partition.

Caching packages allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs when reloading a package; or, in the case of dynamic SQL, eliminating the need for compilation. Sections are kept in the package cache until one of the following occurs:
- The database is shut down
- The package or dynamic SQL statement is invalidated
- The cache runs out of space.

This caching of the section for a static or dynamic SQL statement can improve performance especially when the same statement is used multiple times by applications connected to a database. This is particularly important in a transaction processing application.

By taking the default (-1) in a server or partitioned database environment, the value used to calculate the page allocation is eight times the value specified for the *maxappls* configuration parameter. The exception to this occurs if eight times *maxappls* is less than 32. In this situation, the default value of -1 will set *pckcachesz* to 32.

Recommendation: When tuning this parameter, you should consider whether the extra memory being reserved for the package cache might be more effective if it was allocated for another purpose, such as the buffer pool. For this reason, you should use benchmarking techniques when tuning this parameter.

Tuning this parameter is particularly important when several sections are used initially and then only a few are run repeatedly. If the cache is too large, memory is wasted holding copies of the initial sections.

See the *System Monitor Guide and Reference* for information about the following monitor elements:
- *pkg_cache_lookups* (package cache lookups)
- *pkg_cache_inserts* (package cache inserts)
- *pkg_cache_size_top* (largest package cache size)
- *pkg_cache_num_overflows* (number of package cache overflows)

These database system monitor elements can help you determine whether you should adjust this configuration parameter.

Note: The package cache is a working cache, so you cannot set this parameter to zero. There must be sufficient memory allocated in this cache to hold all sections of the SQL statements currently being executed. If there is more space allocated than currently needed, then sections are cached. These sections can simply be executed the next time they are needed without having to load or compile them.

The limit specified by the *pckcachesz* parameter is a soft limit. This limit may be exceeded, if required, if memory is still available in the database shared set. You can use the *pkg_cache_size_top* monitor element to determine the largest that the package cache has grown, and the *pkg_cache_num_overflows* monitor element to determine how many times the limit specified by the *pckcachesz* parameter has been exceeded.

Application Shared Memory

The following parameter specifies the work area that is used by all agents (both coordinating and subagents) that work for an application:

- "Application Control Heap Size (app_ctl_heap_sz)"

Application Control Heap Size (app_ctl_heap_sz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	**Database server with local and remote clients** 128 [1–64 000] **Database server with local clients** 64 [1–64 000] (for non-UNIX platforms) 128 [1–64 000] (for UNIX-based platforms) **Partitioned database server with local and remote clients** 256 [1–64 000]
Unit of Measure	Pages (4 KB)
When Allocated	When an application starts
When Freed	When an application completes
Related Parameters	• "Catalog Cache Size (catalogcache_sz)" on page 331 • "Application Heap Size (applheapsz)" on page 345 • "Enable Intra-Partition Parallelism (intra_parallel)" on page 445

This parameter determines the maximum size, in 4 KB pages, for the application control shared memory. Application control heaps are allocated from this shared memory.

One application control heap is allocated for each application at the database where the application is active (or, in the case of a partitioned database system, at each database partitionwhere the application is active). The heap is allocated during connect processing by the first agent to receive a request for the application at the database (or database partition). The heap is required to share information between agents working on behalf of the same application (in a partitioned database environment, the sharing occurs at the database partition level: sharing does not occur across database partitions).

This heap is also used to store descriptor information for declared temporary tables. The descriptor information for all declared temporary tables that have not been explicitly dropped is kept in this heap's memory and cannot be dropped until the declared temporary table is dropped.

Notes:

1. In a partitioned database environment, this heap is used to store copies of the executing sections of SQL statements for agents and subagents. Symmetric multiprocessor agents (SMP) subagents, however, use *applheapsz*, as do agents in all other environments.
2. Allocation only occurs for other databases that have the *intra_parallel* parameter set on, and the CURRENT DEGREE special register set to a value greater than one (1). For more information about the CURRENT DEGREE special register, refer to the *SQL Reference*.

Recommendation: Initially, start with the default value. You may have to set the value higher if you are running complex applications, if you have a system that contains a large number of database partitions, or if you use declared temporary tables. The amount of memory needed increases with the number of concurrently active declared temporary tables. A declared temporary table with many columns has a larger table descriptor size than a table with few columns, so having a large number of columns in an application's declared temporary tables also increases the demand on the application control heap.

Agent Private Memory

The following parameters affect the amount of memory used for each database agent:

- "Sort Heap Size (sortheap)" on page 342.
- "Sort Heap Threshold (sheapthres)" on page 342.
- "Statement Heap Size (stmtheap)" on page 344.
- "Application Heap Size (applheapsz)" on page 345.
- "Statistics Heap Size (stat_heap_sz)" on page 345.
- "Query Heap Size (query_heap_sz)" on page 346.
- "DRDA Heap Size (drda_heap_sz)" on page 347.
- "UDF Shared Memory Set Size (udf_mem_sz)" on page 348.
- "Agent Stack Size (agent_stack_sz)" on page 349.
- "Minimum Committed Private Memory (min_priv_mem)" on page 351.
- "Private Memory Threshold (priv_mem_thresh)" on page 351.
- "Maximum Java Interpreter Heap Size (java_heap_sz)" on page 362. On UNIX-based platforms, *java_heap_sz* is allocated per agent.

See “How DB2 Uses Memory” on page 227 for information about how the private agent memory relates to the rest of the memory allocated by the database manager.

Sort Heap Size (sortheap)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	256 [16 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	As needed to perform sorts
When Freed	When sorting is complete
Related Parameters	“Sort Heap Threshold (sheapthres)”

This parameter defines the maximum number of private memory pages to be used for private sorts, or the maximum number of shared memory pages to be used for shared sorts. If the sort is a private sort, then this parameter affects agent private memory. If the sort is a shared sort, then this parameter affects the database shared memory. Each sort has a separate sort heap that is allocated as needed, by the database manager. This sort heap is the area where data is sorted. If directed by the optimizer, a smaller sort heap than the one specified by this parameter is allocated using information provided by the optimizer.

Recommendation:
- Appropriate indexes can minimize the use of the sort heap.
- Increase the size of this parameter when frequent large sorts are required.
- When increasing the value of this parameter, you should examine whether the *sheapthres* parameter in the database manager configuration file also needs to be adjusted.
- The sort heap size is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Sort Heap Threshold (sheapthres)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients

Parameter Type	Configurable
Default [Range]	**UNIX 32-bit platforms** 20 000 [250 — 2 097 152]
	UNIX 64-bit platforms 20 000 [250 — 2 147 483 647]
	OS/2 and Windows NT 10 000 [250 — 2 097 152]
Unit of Measure	Pages (4 KB)
Related Parameters	"Sort Heap Size (sortheap)" on page 342

Private and shared sorts use memory from two different memory sources. The size of the shared sort memory area is statically predetermined at the time of the first connection to a database based on the value of *sheapthres*. The size of the private sort memory area is unrestricted.

The *sheapthres* parameter is used differently for private and shared sorts:

- For private sorts, this parameter is an instance-wide *soft* limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private-sort memory consumption for an instance reaches this limit, the memory allocated for additional incoming private-sort requests will be considerably reduced.
- For shared sorts, this parameter is a database-wide hard limit on the total amount of memory consumed by shared sorts at any given time. When this limit is reached, no further shared-sort memory requests will be allowed (until the total shared-sort memory consumption falls below the limit specified by *sheapthres*).

Examples of those operations that use the sort heap include: hash joins and operations where the table is in memory.

Explicit definition of the threshold prevents the database manager from using excessive amounts of memory for large numbers of sorts.

Recommendation: Ideally, you should set this parameter to a reasonable multiple of the largest *sortheap* parameter you have in your database manager instance. This parameter should be **at least** two times the largest *sortheap* defined for any database within the instance.

If you are doing private sorts and your system is not memory constrained, an ideal value for this parameter can be calculated using the following steps:

1. Calculate the typical sort heap usage for each database:

```
        (typical number of concurrent agents running against the database)
      * (sortheap, as defined for that database)
```

2. Calculate the sum of the above results, which provides the total sort heap that could be used under typical circumstances for all databases within the instance.

For information about performing sorts in an SMP environment, see "Parallel Sort Strategies" on page 183.

You should use benchmarking techniques to tune this parameter to find the proper balance between sort performance and memory usage. See "Chapter 12. Benchmark Testing" on page 299 for more information. Also see "Sorting" on page 248 for more information on sorting.

You can use the database system monitor to track the sort activity.

For more information refer to the following monitor element description in the *System Monitor Guide and Reference*:
- *post_threshold_sorts (post threshold sorts)*

Statement Heap Size (stmtheap)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	2048 [128 – 60 000]
Unit of Measure	Pages (4 KB)
When Allocated	For each statement during precompiling or binding
When Freed	When precompiling or binding of each statement is complete

The statement heap is used as a work space for the SQL compiler during compilation of an SQL statement. This parameter specifies the size of this work space.

This area does not stay permanently allocated, but is allocated and released for every SQL statement handled. Note that for dynamic SQL statements, this work area will be used during execution of your program; whereas, for static SQL statements, it is used during the bind process but not during program execution.

Recommendation: In most cases the default value of this parameter will be acceptable. If you have very large SQL statements and the database manager issues an error (that the statement is too complex) when it attempts to optimize a statement, you should increase the value of this parameter in regular increments (such as 256 or 1024) until the error situation is resolved.

Application Heap Size (applheapsz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	128 [16 – 60 000]
	64 [16 – 60 000] (multipartition)
Unit of Measure	Pages (4 KB)
When Allocated	When an agent is initialized to do work for an application
When Freed	When an agent completes the work to be done for an application
Related Parameters	"Application Control Heap Size (app_ctl_heap_sz)" on page 340

This parameter defines the number of private memory pages available to be used by the database manager on behalf of a specific agent or subagent.

The heap is allocated when an agent or subagent is initialized for an application. The amount allocated will be the minimum amount needed to process the request given to the agent or subagent. As the agent or subagent requires more heap space to process larger SQL statements, the database manager will allocate memory as needed, up to the maximum specified by this parameter.

Note: In a partitioned database environment, the application control heap (*app_ctl_heap_sz*) is used to store copies of the executing sections of SQL statements for agents and subagents. SMP subagents, however, use *applheapsz*, as do agents in all other environments.

Recommendation: Increase the value of this parameter if your applications receive an error indicating that there is not enough storage in the application heap.

The application heap (*applheapsz*) is allocated out of agent private memory.

Statistics Heap Size (stat_heap_sz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	4384 [1096 – 524 288]

Unit of Measure	Pages (4 KB)
When Allocated	When the RUNSTATS utility is started
When Freed	When the RUNSTATS utility is completed
Related Parameters	• "Number of Frequent Values Retained (num_freqvalues)" on page 422 • "Number of Quantiles for Columns (num_quantiles)" on page 423

This parameter indicates the **maximum** size of the heap used in collecting statistics using the RUNSTATS command.

Recommendation: The default value is appropriate when no distribution statistics are collected or when distribution statistics are only being collected for relatively narrow tables. The minimum value is **not** recommended when distribution statistics are being gathered, as only tables containing 1 or 2 columns will fit in the heap.

You should adjust this parameter based on the number of columns for which statistics are being collected. Narrow tables, with relatively few columns, require less memory for distribution statistics to be gathered. Wide tables, with many columns, require significantly more memory. If you are gathering distribution statistics for tables which are very wide and require a large statistics heap, you may wish to collect the statistics during a period of low system activity so you do not interfere with the memory requirements of other users.

Query Heap Size (query_heap_sz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	1000 [2 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When an application (either local or remote) connects to the database

When Freed	When the application disconnects from the database, or detaches from the instance
Related Parameters	"Application Support Layer Heap Size (aslheapsz)" on page 353

This parameter specifies the **maximum** amount of memory that can be allocated for the query heap. A query heap is used to store each query in the agent's private memory. The information for each query consists of the input and output SQLDA, the statement text, the SQLCA, the package name, creator, section number, and consistency token. This parameter is provided to ensure that an application does not consume unnecessarily large amounts of virtual memory within an agent.

The query heap is also used as the source of memory for the memory allocated for blocking cursors. This memory consists of a cursor control block and a fully resolved output SQLDA.

The initial query heap allocated will be the same size as the application support layer heap, as specified by the *aslheapsz* parameter. The query heap size must be greater than or equal to two (2), and must be greater than or equal to the *aslheapsz* parameter. If this query heap is not large enough to handle a given request, it will be reallocated to the size required by the request (not exceeding *query_heap_sz*). If this new query heap is more than 1.5 times larger than *aslheapsz*, the query heap will be reallocated to the size of *aslheapsz* when the query ends.

Recommendation: In most cases the default value will be sufficient. As a minimum, you should set *query_heap_sz* to a value at least five times larger than *aslheapsz*. This will allow for queries larger than *aslheapsz* and provide additional memory for three or four blocking cursors to be open at a given time.

If you have very large LOBs, you may need to increase the value of this parameter so the query heap will be large enough to accommodate those LOBs.

DRDA Heap Size (drda_heap_sz)

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

	• Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	128 [16 – 60 000]
Unit of Measure	Pages (4 KB)
When Allocated	• The DRDA Application Server (AS) allocates a DRDA heap each time a DRDA Application Requester (AR) connects to a DB2 database • DB2 Connect allocates a DRDA heap each time it connects to a DRDA AS.
When Freed	When a DRDA AR disconnects from the database

This parameter indicates the number of pages to allocate for the memory used by DB2 Connect and the DRDA Application Server Support Feature. The following items affect the amount of memory allocated out of this heap:
- The number of cursors opened by an application
- The number of input host variables
- The number of items in the select list
- The size of input and output data
- The length of SQL statements being bound or prepared.

Recommendation: Use the default value unless you receive an error code indicating that you do not have enough DRDA heap.

UDF Shared Memory Set Size (udf_mem_sz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	256 [128 – 60 000]
Unit of Measure	Pages (4 KB)
When Allocated	When a UDF starts
When Freed	When a UDF completes

This parameter is common to both fenced and unfenced User Defined Functions (UDFs). For a fenced UDF, it specifies the default allocation for memory to be shared between the database process and the UDF. In a single-partition database environment, there is only one shared memory set. In a partitioned database environment, there is a shared memory set for each database partition server, and all application agents and sub-agents running on that server use the same shared memory set.

For an unfenced UDF it specifies the size of the private memory set. In a single-partition database environment, the heap is allocated from private memory. In a partitioned database environment, the heap is allocated from the Application Global memory for each database partition server and all agents and subagents running on behalf of the application on that database partition server use the same shared memory set.

For both fenced and unfenced UDFs, this memory is used to pass data to a UDF and back to a database.

If no UDFs are used in applications, the memory is not allocated. If both fenced and unfenced UDFs are running in the same application, two memory allocations result: one for fenced UDFs, and one for unfenced UDFs.

For more information about user-defined functions, refer to the *Application Development Guide* and the *SQL Reference*.

Recommendation: The default setting should be adequate for all cases not involving the passing of LOB data to a UDF. For cases which pass LOB data to a UDF, you may need to increase the amount of memory allocated. You should set the value of this parameter at least 2 pages larger than the size of the input arguments and the result of the external function.

Note: The memory requirement for UDFs tends to be additive, so the number of UDFs referenced in an application will affect the optimal setting for this parameter.

Agent Stack Size (agent_stack_sz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable

Default [Range]		
	OS/2	64 [8 – 1000]
	Windows NT	16 [8 – 1000]
Unit of Measure	Pages (4 KB)	
When Allocated	When an agent is initialized to do work for an application	
When Freed	When an agent completes the work to be done for an application	

The agent stack is the virtual memory that is allocated by DB2 for each agent. This memory is committed when it is required to process an SQL statement. You can use this parameter to optimize memory utilization of the server for a given set of applications. More complex queries will use more stack space, compared to the space used for simple queries.

This parameter does not apply to UNIX-based platforms.

Recommendation: In most cases you should be able to use the default stack size. Only if your environment includes many highly complex queries should you need to increase the value of this parameter. If the stack size is not large enough to process your SQL statement, an error entry will be logged to the db2diag.log file, and an SQL code will be issued. You need to increase *agent_stack_sz* and restart the database instance.

You may be able to reduce the stack size in order to make more address space available to other clients, if your environment matches the following:
- Contains only simple applications (for example light OLTP), in which there are never complex queries
- Requires a relatively large number of concurrent clients (for example, more than 100).

The agent stack size and the number of concurrent clients are inversely related: a larger stack size reduces the potential number of concurrent clients that can be running. This occurs because address space is limited on the OS/2 and Windows NT platforms. For example, on OS/2, assume that you have 400 MB of address space (though the amount depends on the `config.sys` file). If you set the value for *agent_stack_sz* to 1 MB, you will not be able to get more than 400 agents. (In fact, because of other requirements for address space, such as buffer pools, you will probably get far fewer agents.) This means that if you have set *maxagents* to a larger value (for example, 5000), you will never approach this limit.

Minimum Committed Private Memory (min_priv_mem)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	32 [32 – 112 000]
Unit of Measure	Pages (4 KB)
When Allocated	When the database manager is started
When Freed	When the database manager is stopped
Related Parameters	"Private Memory Threshold (priv_mem_thresh)"

This parameter specifies the number of pages that the database server process will reserve as private virtual memory, when a database manager instance is started (`db2start`). If the server requires more private memory, it will try to obtain more from the operating system when required.

This parameter does not apply to UNIX-based systems.

Recommendation: Use the default value.

You should only change the value of this parameter if you want to commit more memory to the database server. This action will save on allocation time. You should be careful, however, that you do not set that value too high, as it can impact the performance of non-DB2 applications.

Private Memory Threshold (priv_mem_thresh)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable

Default [Range]	1296 [-1; 32 – 112 000]
	32 [-1; 32 – 112 000] on Satellite database server with local clients
Unit of Measurement	Pages (4 KB)
Related Parameters	"Minimum Committed Private Memory (min_priv_mem)" on page 351

This parameter is used to determine the amount of unused agent private memory that will be kept allocated, ready to be used by new agents that are started. It does not apply to UNIX-based platforms.

When an agent is terminated, instead of automatically deallocating all of the memory that was used by that agent, the database manager will only deallocate *excess memory allocations*, which is determined by the following formula:

```
Private memory allocated -
(private memory used + priv_mem_thresh)
```

If this formula produces a negative result, no action will be taken.

The following table provides an example to illustrate when memory will be allocated and deallocated. This example uses 100 as an arbitrary setting for *priv_mem_thresh*.

Description of Action	**Memory Allocated**	**Memory Used**
A number of agents are running and have allocated memory.	1000	1000
A new agent is started and uses 100 pages of memory.	1100	1100
A agent using 200 pages of memory terminates. (Notice that 100 pages of memory is freed, while 100 pages is kept allocated for future possible use.)	1000	900
A agent using 50 pages of memory terminates. (Notice that 50 pages of memory is freed and 100 extra pages are still allocated, compared to what is being used by the existing agents.)	950	850
A new agent is started and requires 150 pages of memory. (100 of the 150 pages are already allocated and the database manager only needs to allocate 50 additional pages for this agent.)	1000	1000

A value of “-1”, will cause this parameter to use the value of the *min_priv_mem* parameter.

Recommendation: When setting this parameter, you should consider the client connection/disconnection patterns as well as the memory requirements of other processes on the same machine.

If there is only a brief period during which many clients are concurrently connected to the database, a high threshold will prevent unused memory from being decommitted and made available to other processes. This case results in poor memory management which can affect other processes which require memory.

If the number of concurrent clients is more uniform and there are frequent fluctuations in this number, a high threshold will help to ensure memory is available for the client processes and reduce the overhead to allocate and deallocate memory.

Agent/Application Communication Memory

The following parameters affect the amount of memory that is allocated to allow data to be passed between your application and agent processes:

- “Application Support Layer Heap Size (aslheapsz)”
- “Client I/O Block Size (rqrioblk)” on page 355
- “DOS Requester I/O Block Size (dos_rqrioblk)” on page 356

See “How DB2 Uses Memory” on page 227 for information about how this agent/application shared memory relates to the rest of the memory allocated by the database manager.

Application Support Layer Heap Size (aslheapsz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	15 [1 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the database manager agent process is started for the local application

When Freed	When the database manager agent process is terminated
Related Parameters	"Query Heap Size (query_heap_sz)" on page 346

The application support layer heap represents a communication buffer between the local application and its associated agent. This buffer is allocated as shared memory by each database manager agent that is started.

If the request to the database manager, or its associated reply, do not fit into the buffer they will be split into two or more send-and-receive pairs. The size of this buffer should be set to handle the majority of requests using a single send-and-receive pair. The size of the request is based on the storage required to hold:
- The input SQLDA
- All of the associated data in the SQLVARs
- The output SQLDA
- Other fields which do not generally exceed 250 bytes.

In addition to this communication buffer, this parameter is also used to determine the I/O block size when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

The data sent from the local application is received by the database manager into a set of contiguous memory allocated from the query heap. The *aslheapsz* parameter is used to determine the initial size of the query heap (for both local and remote clients). The maximum size of the query heap is defined by the *query_heap_sz* parameter.

Recommendation: If your application's requests are generally small and the application is running on a memory constrained system, you may wish to reduce the value of this parameter. If your queries are generally very large, requiring more than one send and receive request, and your system is not constrained by memory, you may wish to increase the value of this parameter.

Use the following formula to calculate the number of pages for *aslheapsz*:

```
   aslheapsz >= ( sizeof(input SQLDA)
                + sizeof(each input SQLVAR)
                + sizeof(output SQLDA)
                + 250 ) / 4096
```

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks may also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application. For more information about the OPTIMIZE FOR clause, see "OPTIMIZE FOR n ROWS Clause" on page 73.

Client I/O Block Size (rqrioblk)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	32 767 [4096 – 65 535]
Unit of Measure	Bytes
When Allocated	• When a remote client application issues a connection request for a server database • When a blocking cursor is opened, additional blocks are opened at the client
When Freed	• When the remote application disconnects from the server database • When the blocking cursor is closed
Related Parameters	"DOS Requester I/O Block Size (dos_rqrioblk)" on page 356

This parameter specifies the size of the communication buffer between remote applications and their database agents on the database server. When a database client requests a connection to a remote database, this communication buffer is allocated on the client. On the database server, a communication buffer of 32767 bytes is initially allocated, until a connection is established and the server can determine the value of *rqrioblk* at the client. Once the server knows this value, it will reallocate its communication buffer if the client's buffer is not 32767 bytes.

In addition to this communication buffer, this parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

Recommendation: For non-blocking cursors, a reason for increasing the value of this parameter would be if the data (for example, large object data) to be transmitted by a single SQL statement is so large that the default value is insufficient.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks may also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application. For more information on the OPTIMIZE FOR clause, see "OPTIMIZE FOR n ROWS Clause" on page 73.

DOS Requester I/O Block Size (dos_rqrioblk)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	4096 [4096 – 65 535]

Unit of Measurement	Bytes
When Allocated	• When a remote DOS or Windows 3.1 client issues a connection request to a server database • When a blocking cursor is opened, additional blocks are opened at the client
When Freed	• When the remote application disconnects from the database • When a blocking cursor is closed
Related Parameters	"Client I/O Block Size (rqrioblk)" on page 355

This parameter specifies the size of the communication buffer between DOS/Windows 3.1 applications and their database agents on the database server. This parameter is similar to the *rqrioblk* parameter, except it allows you to set a different value for blocks used with DOS/Windows 3.1 clients. In a DB2 configuration file, you can set both the *rqrioblk* parameter (used for Windows 32-bit, OS/2, and UNIX clients) and the *dos_rqrioblk* parameter (used for DOS and Windows 3.1 clients).

In addition to this communication buffer, this parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

Recommendation: For non-blocking cursors, a reason for increasing the value of this parameter would be if the data (for example, large object data) to be transmitted by a single SQL statement is so large that the default value is insufficient.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks may also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application. For more information on the OPTIMIZE FOR clause, see "OPTIMIZE FOR n ROWS Clause" on page 73.

Database Manager Instance Memory

The following parameters affect memory that is allocated and used at an instance level:

- "Database System Monitor Heap Size (mon_heap_sz)"
- "Directory Cache Support (dir_cache)" on page 359
- "Audit Buffer Size (audit_buf_sz)" on page 361
- "Maximum Java Interpreter Heap Size (java_heap_sz)" on page 362

Database System Monitor Heap Size (mon_heap_sz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	**UNIX** 56 [0 – 60 000] **OS/2 and Windows NT Database server with local and remote clients and Satellite database server with local clients** 24 [0 – 60 000] **OS/2 and Windows NT Database server with local clients** 12 [0 – 60 000]
Unit of Measure	Pages (4 KB)
When Allocated	When the database manager is started with the *db2start* command
When Freed	When the database manager is stopped with the *db2stop* command
Related Parameters	"Default Database System Monitor Switches (dft_monswitches)" on page 448

This parameter determines the amount of the memory, in pages, to allocate for database system monitor data. Memory is allocated from the monitor heap when you perform database monitoring activities such as taking a snapshot, turning on a monitor switch, resetting a monitor, or activating an event monitor.

A value of zero prevents the database manager from collecting database system monitor data.

Recommendation: The amount of memory required for monitoring activity depends on the number of monitoring applications (applications taking snapshots or event monitors), which switches are set, and the level of database activity.

The following formula provides an approximation of the number of pages required for the monitor heap:

```
( number of monitoring applications + 1 ) *
( number of databases *
  (800 + ( number of tables accessed * 20 )
   + ( ( number of applications connected + 1) *
       (200 + (number of table spaces * 100) ) ) ) )
/ 4096
```

If the available memory in this heap runs out, one of the following will occur:
- A level 2 error message is written to the `db2alert.log` and `db2diag.log` files, when the first application connects to the database for which this event monitor is defined.
- An error code is returned to your application, if an event monitor being started dynamically using the SET EVENT MONITOR statement fails.
- An error code is returned to your application, if a monitor command or API subroutine fails.

Directory Cache Support (dir_cache)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	Yes [Yes; No]
When Allocated	

- When an application issues its first connect, the private cache is allocated
- When a database manager instance is started (`db2start`), the shared cache is allocated.

When Freed

- When an the application process terminates, the private cache is freed
- When a database manager instance is stopped (`db2stop`), the shared cache is freed.

By setting *dir_cache* to “yes” the database, node and DCS directory files will be cached in memory. The use of the directory cache reduces connect costs by eliminating directory file I/O and minimizing the directory searches required to retrieve directory information. There are two types of directory caches:
- A private cache that is allocated and used for each application process, on the machine at which the application is running.
- A shared cache that is allocated and used for some of the internal database manager processes.

Note: Only the private cache is applicable to supported Windows environments.

For private caches, when an application issues its first connect, each directory file is read and the information is cached in private memory for this application. The cache is used by the application process on subsequent connect requests and is maintained for the life of the application process. If a database is not found in the private cache, the directory files are searched for the information, but the cache is not updated. If the application modifies a directory entry, the next connect within that application will cause the cache for this application to be refreshed. The private cache for other applications will not be refreshed. When the application process terminates, the cache is freed. (To refresh the directory cache used by a command line processor session, issue a `db2 terminate` command.)

For shared caches, when a database manager instance is started (`db2start`), each directory file is read and the information is cached in shared memory. This cache is used by some of the database manager processes and is maintained until the instance is stopped (`db2stop`). If a directory entry is not found in this cache, the directory files are searched for the information. This shared cache is never refreshed during the time the instance is running.

Recommendation: Use directory caching if your directory files do not change frequently and performance is critical.

In addition, on remote clients, directory caching can be beneficial if your applications issue several different connection requests. In this case, caching reduces the number of times a single application must read the directory files.

Directory caching can also improve the performance of taking database system monitor snapshots. In addition, you should explicitly reference the database name on the snapshot call, instead of using database aliases.

Note: Errors may occur when performing snapshot calls if directory caching is turned on and if databases are cataloged, uncataloged, created, or dropped after the database manager is started.

Audit Buffer Size (audit_buf_sz)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	0 [0 – 65 000]
Unit of Measure	Pages (4 KB)
When Allocated	When DB2 is started
When Freed	When DB2 is stopped

This parameter specifies the size of the buffer used when auditing the database. For more information about the audit facility, refer to “Auditing DB2 Activities” in *Administration Guide: Planning*.

The default value for this parameter is zero (0). If the value is zero (0), the audit buffer is not used. If the value is greater than zero (0), space is allocated for the audit buffer where the audit records will be placed when they are generated by the audit facility. The value times 4 KB pages is the amount of space allocated for the audit buffer. The audit buffer cannot be allocated dynamically; DB2 must be stopped and then restarted before the new value for this parameter takes effect.

By changing this parameter from the default to some value larger than zero (0), the audit facility writes records to disk asynchronously compared to the execution of the statements generating the audit records. This improves DB2 performance over leaving the parameter value at zero (0). The value of zero

(0) means the audit facility writes records to disk synchronously with (at the same time as) the execution of the statements generating the audit records. The synchronous operation during auditing decreases the performance of applications running in DB2.

Maximum Java Interpreter Heap Size (java_heap_sz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	512 [0 - 4 096]
Unit of Measure	Pages (4 KB)
When Allocated	When a Java application starts
When Freed	When a Java application completes
Related Parameters	"Java Development Kit 1.1 Installation Path (jdk11_path)" on page 456

This parameter determines the maximum size of the heap that is used by the Java interpreter.

There is one heap for each DB2 process (one for each agent or subagent on UNIX-based platforms, and one for each instance in other platforms), and there is also one heap for each fenced UDF and fenced stored procedure process. In all situations, only the agents or processes that run Java UDFs or stored procedures ever allocate this memory. On partitioned database systems, the heap is multiplied by the number of database partition servers.

Locks

The following parameters influence how locking is managed in your environment:
- "Time Interval for Checking Deadlock (dlchktime)" on page 363
- "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 364
- "Lock Timeout (locktimeout)" on page 365

See also “Maximum Storage for Lock List (locklist)” on page 335.

“Locking” on page 48 provides a general overview of how the database manager uses locking to maintain data integrity.

Time Interval for Checking Deadlock (dlchktime)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	10 000 (10 seconds) [1000 – 600 000]
Unit of Measure	Milliseconds
Related Parameters	• “Maximum Storage for Lock List (locklist)” on page 335 • “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 364

A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting is never resolved because each application is holding a resource that the other needs to continue.

The deadlock check interval defines the frequency at which the database manager checks for deadlocks among all the applications connected to a database.

Notes:

1. In a partitioned database environment, this parameter applies to the catalog node only.
2. In a partitioned database environment, a deadlock is not flagged until after the second iteration.

Recommendation: Increasing this parameter decreases the frequency of checking for deadlocks, thereby increasing the time that application programs must wait for the deadlock to be resolved.

Decreasing this parameter increases the frequency of checking for deadlocks, thereby decreasing the time that application programs must wait for the deadlock to be resolved but increasing the time that the database manager takes to check for deadlocks. If the deadlock interval is too small, it can decrease run-time performance, because the database manager is frequently performing deadlock detection. If this parameter is set lower to improve concurrency, you should ensure that *maxlocks* and *locklist* are set appropriately to avoid unnecessary lock escalation, which can result more lock contention and as a result, more deadlock situations.

Maximum Percent of Lock List Before Escalation (maxlocks)

Configuration Type	Database	
Parameter Type	Configurable	
Default [Range]		
	UNIX	10 [1 – 100]
	OS/2 and Windows NT	22 [1 – 100]
Unit of Measure	Percentage	
Related Parameters	• "Maximum Storage for Lock List (locklist)" on page 335 • "Maximum Number of Active Applications (maxappls)" on page 374	

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. This parameter defines a percentage of the lock list held by an application that must be filled before the database managerperforms escalation. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the *maxlocks* value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of *maxlocks*. The *maxlocks* parameter multiplied by the *maxappls* parameter cannot be less than 100.

Recommendation: When setting *maxlocks*, you should consider the size of the lock list (*locklist*):

```
maxlocks = 100 *
           (512 locks per application
           * 32 bytes per lock
           * 2) / (locklist * 4096 bytes)
```

This sample formula allows any application to hold twice the average number of locks.

You can increase *maxlocks* if few applications run concurrently since there will not be a lot of contention for the lock list space in this situation.

You may use the database system monitor to help you track and tune this configuration parameter.

For more information see the *locks_held_top (maximum number of locks held)* monitor element description in the *System Monitor Guide and Reference*.

The control of lock escalation through this parameter is important to the optimizer since it uses this parameter to determine access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Lock Timeout (locktimeout)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	-1 [-1; 0 – 30 000]
Unit of Measurement	Seconds
Related Parameters	• "Maximum Storage for Lock List (locklist)" on page 335 • "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 364

This parameter specifies the number of seconds that an application will wait to obtain a lock. This helps avoid global deadlocks for applications.

If you set this parameter to `0`, locks are not waited for. In this situation, if no lock is available at the time of the request, the application immediately receives a -911.

If you set this parameter to `-1`, lock timeout detection is turned off. In this situation a lock will be waited for (if one is not available at the time of the request) until either of the following:
- The lock is granted
- A deadlock occurs.

Recommendation: In a transaction processing (OLTP) environment, you can use an initial starting value of 30 seconds. In a query-only environment you could start with a higher value. In both cases, you should use benchmarking techniques to tune this parameter.

When working with DataLinks Manager, if you see lock timeouts in the db2diag.log of the DataLinks Manager (dlfm) instance, then you should increase the value of *locktimeout*. You should also consider increasing the value of *locklist*.

The value should be set to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of a user leaving their workstation). You should set it high enough so valid lock requests do not time-out because of peak workloads, during which time, there is more waiting for locks.

You may use the database system monitor to help you track the number of times an application (connection) experienced a lock timeout or that a database detected a timeout situation for all applications that were connected. For more information see the *locks_timeouts* (number of lock timeouts) monitor element description in the *System Monitor Guide and Reference*.

High values of the *lock_timeout* monitor element can be caused by:
- Too low a value for this configuration parameter.
- An application (transaction) that is holding lock(s) for an extended period. You can use the database system monitor to further investigate these applications.
- A concurrency problem, that could be caused by lock escalations (from the row-level to a table-level). See “Maximum Percent of Lock List Before Escalation (maxlocks)” on page 364 and “Maximum Storage for Lock List (locklist)” on page 335 for more information.

For more information on the use of this parameter see “Lock Waits and Timeouts” on page 55.

I/O and Storage

The following parameters can influence I/O and storage costs related to the operation of your database:
- “Changed Pages Threshold (chngpgs_thresh)”
- “Number of Asynchronous Page Cleaners (num_iocleaners)” on page 367
- “Number of I/O Servers (num_ioservers)” on page 369
- “Index Sort Flag (indexsort)” on page 369
- “Sequential Detection Flag (seqdetect)” on page 370
- “Default Prefetch Size (dft_prefetch_sz)” on page 370
- “Default Number of SMS Containers (numsegs)” on page 371
- “Default Extent Size of Table Spaces (dft_extent_sz)” on page 372
- “Extended Storage Memory Segment Size (estore_seg_sz)” on page 372
- “Number of Extended Storage Memory Segments (num_estore_segs)” on page 373

Changed Pages Threshold (chngpgs_thresh)

Configuration Type	Database
Parameter Type	Configurable

Default [Range]	60 [5 – 99]
Unit of Measure	Percentage
Related Parameters	"Number of Asynchronous Page Cleaners (num_iocleaners)"

Asynchronous page cleaners will write changed pages from the buffer pool (or the buffer pools) to disk before the space in the buffer pool is required by a database agent. This means that the agents will not wait for a changed page to be written out, before being able to read a page, and your application's transactions should run faster.

You may use this parameter to specify the level (percentage) of changed pages at which the asynchronous page cleaners will be started, if they are not currently active. When the page cleaners are started, they will build a list of the pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start.

In a read-only (for example, query) environment, these page cleaners are not used.

Recommendation: For databases with a heavy update transaction workload, you can generally ensure that there are enough clean pages in the buffer pool by setting the parameter value to be equal-to or less-than the default value. A percentage larger than the default can help performance if your database has a small number of very large tables.

Number of Asynchronous Page Cleaners (num_iocleaners)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	1 [0 – 255]
Unit of Measure	Counter
Related Parameters	• "Buffer Pool Size (buffpage)" on page 327 • "Changed Pages Threshold (chngpgs_thresh)" on page 366

This parameter allows you to specify the number of asynchronous page cleaners for a database. These page cleaners write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. This means that the agents will not wait for changed pages to be written out, before being able to read a page. As a result, your application's transactions should run faster.

If you set the parameter to zero (0), no page cleaners are started and as a result, the database agents will perform all of the page writes from the buffer pool to disk. This parameter can have a significant performance impact on a database stored across many physical storage devices, since in this case there is a greater chance that one of the devices will be idle. If no page cleaners are configured, your applications may encounter periodic log full conditions.

If the applications for a database primarily consist of transactions that update data, an increase in the number of cleaners will speed up performance. Increasing the page cleaners will also decrease recovery time from soft failures, such as power outages, because the contents of the database on disk will be more up-to-date at any given time.

Recommendation: Consider the following factors when setting the value for this parameter:
- Application type
  - If it is a query-only database that will not have updates, set this parameter to be zero (0). The exception would be if the query work load results in many TEMP tables being created (you can determine this by using the explain utility).
  - If transactions are run against the database, set this parameter to be between one and the number of physical storage devices used for the database.
- Workload

  Environments with high update transaction rates may require more page cleaners to be configured.
- Buffer pool sizes (*buffpage*)

  Environments with large buffer pools may also require more page cleaners to be configured.

You may use the database system monitor to help you tune this configuration parameter using information from the event monitor about write activity from a buffer pool:
- The parameter can be reduced if both of the following conditions are true:
  - *pool_data_writes* is approximately equal to *pool_async_data_writes*
  - *pool_index_writes* is approximately equal to *pool_async_index_writes*.
- The parameter should be increased if either of the following conditions are true:
  - *pool_data_writes* is much greater than *pool_async_data_writes*
  - *pool_index_writes* is much greater than *pool_async_index_writes*.

For more information see the following monitor elements descriptions in the *System Monitor Guide and Reference*:
- *pool_data_writes* (buffer pool data writes)
- *pool_index_writes* (buffer pool index writes)

- *pool_async_data_writes* (buffer pool asynchronous data writes)
- *pool_async_index_writes* (buffer pool asynchronous index writes).

Number of I/O Servers (num_ioservers)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	3 [1 – 255]
	1 [1 – 255] on Satellite database server with local clients
Unit of Measure	Counter
When Allocated	When an application connects to a database
When Freed	When an application disconnects from a database
Related Parameters	• "Default Prefetch Size (dft_prefetch_sz)" on page 370 • "Sequential Detection Flag (seqdetect)" on page 370

I/O servers are used on behalf of the database agents to perform prefetch I/O and asynchronous I/O by utilities such as backup and restore. This parameter specifies the number of I/O servers for a database. No more than this number of I/Os for prefetching and utilities can be in progress for a database at any time. An I/O server waits while an I/O operation that it initiated is in progress. Non-prefetch I/Os are scheduled directly from the database agents and as a result are not constrained by *num_ioservers*.

Recommendation: In order to fully exploit all the I/O devices in the system, a good value to use is generally one or two more than the number of physical devices on which the database resides. It is better to configure additional I/O servers, since there is minimal overhead associated with each I/O server and any unused I/O servers will remain idle.

For more information, see "Prefetching Data into the Buffer Pool" on page 241 and "Configuring I/O Servers for Prefetching and Parallel I/O" on page 244.

Index Sort Flag (indexsort)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Yes [Yes; No]

This parameter indicates whether sorting of index keys will occur during index creation. Performance of index creation is enhanced by performing a sort first, particularly for indexes with low cluster ratios or cluster factors. Performance of queries can also be better if indexes are created with a sort. The cost of this performance enhancement is the increased disk space required for the sort, which could require twice the amount of space as creating an index without performing an initial sort.

Recommendation: Use the default setting (“Yes”), unless you do not have enough disk space. Note that the disk space required for this sort is approximately equal to the amount of space needed to SELECT the columns of the index from the table with an ORDER BY clause on those columns.

If you have a symmetric multiprocessor (SMP) environment and specify “No” for this parameter, the multiple processing that is possible in an SMP environment is not used during index creation.

Sequential Detection Flag (seqdetect)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Yes [Yes; No]
Related Parameters	“Default Prefetch Size (dft_prefetch_sz)”

The database manager can monitor I/O and if sequential page reading is occurring the database manager can activate I/O prefetching. This type of sequential prefetch is known as *sequential detection*. You may use the *seqdetect* configuration parameter to control whether the database manager should perform sequential detection.

If this parameter is set to “no”, prefetching takes place only if the database manager knows it will be useful, for example table sorts, table scans, or list prefetch.

Recommendation: In most cases, you should use the default value for this parameter. Try turning sequential detection off, only if other tuning efforts were unable to correct serious query performance problems.

Default Prefetch Size (dft_prefetch_sz)

Configuration Type	Database	
Parameter Type	Configurable	
Default [Range]		
	UNIX	32 [0 — 32 767]

	OS/2 and Windows NT 16 [0 — 32 767]
Unit of Measure	Pages
Related Parameters	• "Default Extent Size of Table Spaces (dft_extent_sz)" on page 372 • "Number of I/O Servers (num_ioservers)" on page 369

When a table space is created, `PREFETCHSIZE n` can be optionally specified, where `n` is the number of pages the database manager will read if prefetching is being performed. If you do not specify the prefetch size on the CREATE TABLESPACE statement, the database manager uses the value given by this parameter.

For more information, see "Prefetching Data into the Buffer Pool" on page 241.

Recommendation: Using system monitoring tools, you can determine if your CPU is idle while the system is waiting for I/O. Increasing the value of this parameter may help if the table spaces being used do not have a prefetch size defined for them.

This parameter provides the default for the entire database, and it may not be suitable for all table spaces within the database. For example, a value of 32 may be suitable for a table space with an extent size of 32 pages, but not suitable for a table space with an extent size of 25 pages. Ideally, you should explicitly set the prefetch size for each table space.

To help minimize I/O for table spaces defined with the default extent size (*dft_extent_sz*), you should set this parameter as a factor or whole multiple of the value of the *dft_extent_sz* parameter. For example, if the *dft_extent_sz* parameter is 32, you could set *dft_prefetch_sz* to 16 (a factor of 32) or to 64 (a whole multiple of 32). If the prefetch size is a multiple of the extent size, the database manager may perform I/O in parallel, if the following conditions are true:
- The extents being prefetched are on different physical devices
- Multiple I/O servers are configured (*num_ioservers*).

Default Number of SMS Containers (numsegs)

Configuration Type	Database
Parameter Type	Informational
Unit of Measure	Counter

This parameter, which only applies to SMS table spaces, indicates the number of containers that will be created within the default table spaces. This parameter will show the information used when you created your database, whether it was specified explicitly or implicitly on the CREATE DATABASE command. The CREATE TABLESPACE statement **does not** use this parameter in any way.

Refer to "Database Physical Directories" in the *Administration Guide: Planning* for more information.

Default Extent Size of Table Spaces (dft_extent_sz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	32 [2 – 256]
Unit of Measure	Pages
Related Parameters	"Default Prefetch Size (dft_prefetch_sz)" on page 370

When a table space is created, `EXTENTSIZE n` can be optionally specified, where `n` is the extent size. If you do not specify the extent size on the CREATE TABLESPACE statement, the database manager uses the value given by this parameter.

Refer to "Designing and Choosing Table Spaces" in the *Administration Guide: Planning* for more information.

Recommendation: In many cases, you will want to explicitly specify the extent size when you create the table space. Before choosing a value for this parameter, you should understand how you would explicitly choose an extent size for the CREATE TABLESPACE statement. For more information see "Table Space Impact on Query Optimization" on page 90.

Extended Storage Memory Segment Size (estore_seg_sz)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	16 000 [0 – 1 048 575]
Unit of Measure	Pages
Related Parameters	"Number of Extended Storage Memory Segments (num_estore_segs)" on page 373

This parameter specifies the number of pages in each of the extended memory segments in the database. There are platform-dependent considerations when setting this configuration parameter.

Recommendation: This parameter only has an effect when extended storage is available, and is used as shown by the *num_estore_segs* parameter. When specifying the number of pages to be used in each extended memory segment, you should also consider the number of extended memory segments by reviewing and modifying the *num_estore_segs* parameter. For more information about extended storage, see "Extending Memory" on page 265.

Number of Extended Storage Memory Segments (num_estore_segs)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	0 [0 – 214 7483 647]
Related Parameters	"Extended Storage Memory Segment Size (estore_seg_sz)" on page 372

This parameter specifies the number of extended storage memory segments available for use by the database.

The default is no extended storage memory segments.

Recommendation: Only use this parameter to establish the use of extended storage memory segments if your platform environment has more memory than the maximum address space and you wish to use this memory. When specifying the number of segments, you should also consider the size of the each of the segments by reviewing and modifying the *estore_seg_sz* parameter.

When both the *num_estore_segs* and *estore_seg_sz* configuration parameters are set, you should specify which buffer pools will use the extended memory through the CREATE/ALTER BUFFERPOOL statements. For more information about extended storage, see "Extending Memory" on page 265.

Agents

The following parameters can influence the number of applications that can be run concurrently and achieve optimal performance:

- "Maximum Number of Active Applications (maxappls)" on page 374
- "Average Number of Active Applications (avg_appls)" on page 375
- "Maximum Database Files Open per Application (maxfilop)" on page 376
- "Maximum Total Files Open per Application (maxtotfilop)" on page 377
- "Priority of Agents (agentpri)" on page 378
- "Maximum Number of Agents (maxagents)" on page 379

- "Maximum Number of Concurrent Agents (maxcagents)" on page 380
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 381
- "Maximum Number of Logical Agents (max_logicagents)" on page 382
- "Agent Pool Size (num_poolagents)" on page 383
- "Initial Number of Agents in Pool (num_initagents)" on page 384

Maximum Number of Active Applications (maxappls)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	

UNIX	40 [1 – 64 000]
OS/2 and Windows NT Database server with local and remote clients	20 [1 – 64 000]
OS/2 and Windows NT Database server with local clients	10 [1 – 64 000]

Unit of Measure	Counter

Related Parameters

- "Maximum Number of Agents (maxagents)" on page 379
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 381
- "Maximum Percent of Lock List Before Escalation (maxlocks)" on page 364
- "Maximum Storage for Lock List (locklist)" on page 335
- "Average Number of Active Applications (avg_appls)" on page 375

This parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

The value of this parameter must be equal to or greater than the sum of the connected applications, plus the number of these same applications that may be concurrently in the process of completing a two-phase commit or rollback. Then add to this sum the anticipated number of indoubt transactions that might exist at any one time. Refer to “Recovering from Problems During Two-Phase Commit” in the *Administration Guide: Planning* for more information on indoubt transactions.

When an application attempts to connect to a database, but *maxappls* has already been reached, an error is returned to the application indicating that the maximum number of applications have been connected to the database.

As more applications use the DataLinks Manager, the value of *maxappls* should be increased. Use the following formula to compute the value you need:

```
   <maxappls> = 5 * (number of nodes) + (peak number of active applications
                      using DataLinks Manager)
```

The maximum supported value for DataLinks Manager is 2 000.

In a partitioned database environment, this is the maximum number of applications that can be concurrently active against a database partition. This parameter limits the number of active applications against the database partition on a database partition server, regardless of whether the server is the coordinator node for the application or not. The catalog node in a partitioned database environment requires a higher value for *maxappls* than is the case for other types of environments because, in the partitioned database environment, every application requires a connection to the catalog node.

Recommendation: Increasing the value of this parameter without lowering the *maxlocks* parameter or increasing the *locklist* parameter could cause you to reach the database limit on locks (*locklist*) rather than the application limit and as a result cause pervasive lock escalation problems.

To a certain extent, the maximum number of applications is also governed by *maxagents*. An application can only connect to the database, if there is an available connection (*maxappls*) as well as an available agent (*maxagents*). In addition, the maximum number of applications is also controlled by the *max_coordagents* configuration parameter, because no new applications (that is, coordinator agents) can be started if *max_coordagents* has been reached.

Average Number of Active Applications (avg_appls)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	1 [1 – maxappls]
Unit of Measure	Counter
Related Parameters	

- "Maximum Number of Active Applications (maxappls)" on page 374

This parameter is used by the SQL optimizer to help estimate how much buffer pool will be available at run-time for the access plan chosen. Increasing this parameter can influence the optimizer to choose an access plan for queries that will be more conservative in its buffer pool usage.

Recommendation: When running DB2 in a multi-user environment, particularly with complex queries and a large buffer pool, you may want the SQL optimizer to know that multiple query users are using your system so that the optimizer should be more conservative in assumptions of buffer pool availability.

When setting this parameter, you should estimate the number of heavy query applications that typically use the database. This estimate should exclude all light OLTP applications. If you have trouble estimating this number, you can multiply the following:
- An average number of all applications running against your database. The database system monitor can provide information about the number of applications at any given time and using a sampling technique, you can calculate an average over a period of time. The information from the database system monitor includes both OLTP and non-OLTP applications.
- Your estimate of the percentage of heavy query applications.

As with adjusting other configuration parameters that affect the optimizer, you should adjust this parameter in small increments. This allows you to minimize path selection differences.

You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Maximum Database Files Open per Application (maxfilop)

Configuration Type	Database	
Parameter Type	Configurable	
Default [Range]		
	UNIX	64 [2 – 1950]
	OS/2 and Windows NT	64 [2 – 32 768]
Unit of Measure	Counter	
Related Parameters		

- "Maximum Total Files Open per Application (maxtotfilop)" on page 377

- “Maximum Number of Active Applications (maxappls)” on page 374

This parameter specifies the maximum number of file handles that can be open for each database agent. If opening a file causes this value to be exceeded, some files in use by this agent are closed. If *maxfilop* is too small, the overhead of opening and closing files so as not to exceed this limit will become excessive and may degrade performance.

Both SMS table spaces and DMS table space file containers are treated as files in the database manager’s interaction with the operating system, and file handles are required. More files are generally used by SMS table spaces compared to the number of containers used for a DMS file table space. Therefore, if you are using SMS table spaces, you will need a larger value for this parameter compared to what you would require for DMS file table spaces.

You can also use this parameter to ensure that the overall total of file handles used by the database manager does not exceed the operating system limit by limiting the number of handles per agent to a specific number; the actual number will vary depending on the number of agents running concurrently.

Maximum Total Files Open per Application (maxtotfilop)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	16 000 [100 – 32 768]
Unit of Measure	Counter
Related Parameters	“Maximum Database Files Open per Application (maxfilop)” on page 376

This parameter defines the maximum number of files that can be opened by all agents and other threads executing in a single database manager instance. If opening a file causes this value to be exceeded, an error is returned to your application.

Note: This parameter does not apply to UNIX-based platforms.

Recommendation: When setting this parameter, you should consider the number of file handles that could be used for each database in the database manager instance. To estimate an upper limit for this parameter:

1. Calculate the maximum number of file handles that could be opened for each database in the instance, using the following formula:
   ```
   maxappls * maxfilop
   ```
2. Calculate the sum of above results and verify that it does not exceed the parameter maximum.

If a new database is created, you should re-evaluate the value for this parameter.

You should also validate the total file handles that may be used on your system does not exceed the system maximum using the following formula:

```
   (sum of maxtotfilop for all instances on machine)
 + (estimate of file handles required by other applications)
 <= 65535
```

Priority of Agents (agentpri)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	**AIX** -1 [41 - 125] **Other UNIX** -1 [41 - 128] **Windows NT** -1 [0 - 6] **OS/2** -1 [200 - 231; 300 - 331; 400 - 431]

This parameter controls the priority given both to all agents, and to other database manager instance processes and threads, by the operating system scheduler. In a partitioned database environment, this also includes both coordinating and parallel agents, the parallel system controllers, and the FCM daemons. This priority determines how CPU time is given to the DB2 processes, agents, and threads relative to the other processes and threads running on the machine. When the parameter is set to -1, no special action is taken and the database manager is scheduled in the normal way that the operating system schedules all processes and threads. When the parameter is set to a value other than -1, the database manager will create its processes and threads with a static priority set to the value of the parameter. Therefore, this parameter allows you to control the priority with which the database manager processes and threads will execute on your machine.

You can use this parameter to increase database manager throughput. The values for setting this parameter are dependent on the operating system on which the database manager is running. For example, in a UNIX-based environment, numerically low values yield high priorities. When the parameter is set to a value between 41 and 125, the database manager creates its agents with a UNIX static priority set to the value of the parameter. This is important in UNIX-based environments because numerically low values yield high priorities for the database manager, but other processes (including applications and users) may experience delays because they cannot obtain enough CPU time. You should balance the setting of this parameter with the other activity expected on the machine.

In an OS/2 environment, higher numeric values yield higher priorities.

Recommendation: The default value should be used initially. This value provides a good compromise between response time to other users/applications and database manager throughput.

If database performance is a concern, you can use benchmarking techniques to determine the optimum setting for this parameter. You should take care when increasing the priority of the database manager because performance of other user processes can be severely degraded especially when the CPU utilization is very high. Increasing the priority of the database manager processes and threads can have significant performance benefits.

Note: If you set this parameter to a non-default value on UNIX-based platforms, you cannot use the governor to alter agent priorities.

Maximum Number of Agents (maxagents)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients

Parameter Type	Configurable
Default [Range]	200 [1 – 64 000] 400 [1 – 64 000] on Partitioned database server with local and remote clients 10 [1 – 64 000] on Satellite database server with local clients
Unit of Measure	Counter
Related Parameters	• "Maximum Number of Active Applications (maxappls)" on page 374 • "Maximum Number of Concurrent Agents (maxcagents)" • "Maximum Number of Coordinating Agents (max_coordagents)" on page 381 • "Maximum Number of DARI Processes (maxdari)" on page 386 • "Minimum Committed Private Memory (min_priv_mem)" on page 351 • "Agent Pool Size (num_poolagents)" on page 383

This parameter indicates the maximum number of database manager agents, whether coordinating agents or subagents, available at any given time to accept application requests. If you want to limit the number of coordinating agents, use the *max_coordagents* parameter.

This parameter can be useful in memory constrained environments to limit the total memory usage of the database manager, because each additional agent requires additional memory.

Recommendation: The value of *maxagents* should be at least the sum of the values for *maxappls* in each database allowed to be accessed concurrently. If the number of databases is greater than the *numdb* parameter, then the safest course is to use the product of *numdb* with the largest value for *maxappls*.

Each additional agent requires some resource overhead that is allocated at the time the database manager is started.

Maximum Number of Concurrent Agents (maxcagents)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients

- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type	Configurable
Default [Range]	-1 (*max_coordagents*) [-1; 1 – *max_coordagents*]
Unit of Measure	Counter

Related Parameters

- "Maximum Number of Active Applications (maxappls)" on page 374
- "Maximum Number of Agents (maxagents)" on page 379
- "Maximum Number of Coordinating Agents (max_coordagents)"

The maximum number of database manager coordinator agents that can be concurrently executing a database manager transaction. This parameter is used to control the load on the system during periods of high simultaneous application activity. For example, you may have a system requiring a large number of connections but with a limited amount of memory to serve those connections. Adjusting this parameter can be useful in such an environment, where a period of high simultaneous activity could cause excessive operating system paging.

This parameter does not limit the number of applications that can have connections to a database. It only limits the number of database manager agents that can be processed concurrently by the database manager at any one time, thereby limiting the usage of system resources during times of peak processing.

A value of −1 indicates that the limit is *max_coordagents*.

Recommendation: In most cases the default value for this parameter will be acceptable. In cases where the high concurrency of applications is causing problems, you can use benchmark testing to tune this parameter to optimize your performance.

Maximum Number of Coordinating Agents (max_coordagents)

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients

	• Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	-1 (*maxagents* — *num_initagents*) [-1, 0 – *maxagents*] For partitioned database environments and environments in which *intra_parallel* is set to “Yes”, the default is *maxagents* - *num_initagents*; otherwise, the default is *maxagents*. This ensures that, in non-partitioned database environments, *max_coordagents* always equals *maxagents*, unless the system is configured for intra-partition parallelism. If you do not have a partitioned database environment, and have not enabled the *intra_parallel* parameter, *max_coordagents* must equal *maxagents*.
Related Parameters	• “Initial Number of Agents in Pool (num_initagents)” on page 384 • “Agent Pool Size (num_poolagents)” on page 383 • “Maximum Number of Agents (maxagents)” on page 379 • “Enable Intra-Partition Parallelism (intra_parallel)” on page 445

This parameter determines the maximum number of coordinating agents that can exist at one time on a server in a partitioned or non-partitioned database environment.

One coordinating agent is acquired for each local or remote application that connects to a database or attaches to an instance. Requests that require an instance attachment include CREATE DATABASE, DROP DATABASE, and Database System Monitor commands.

Maximum Number of Logical Agents (max_logicagents)

Configuration Type	Database manager
Parameter Type	Configurable

Default [Range]	-1 (*max_coordagents*) [-1; *max_coordagents* — 64 000]

This parameter controls the maximum number of applications that can be connected to the instance. Typically, each application is assigned a coordinator agent. An agent facilitates the operations between the application and the database. When the default value for this parameter is used, the concentrator feature is not activated. As a result, each agent operates with its own private memory and shares database manager and database global resources such as the buffer pool with other agents. When the parameter is set to a value greater than the default, the concentrator feature is activated. The intent of the concentrator is to reduce the server resources per client application to a point where a DB2 Connect gateway can handle greater than 10 000 client connections.

A value or -1 indicates that the limit is *max_coordagents*.

Agent Pool Size (num_poolagents)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	-1 [-1, 0 — *maxagents*] Using the default, the value for a server with a non-partitioned database and local clients is the larger of *maxagents*/50 or *max_querydegree*. Using the default, the value for a server with a non-partitioned database and local and remote clients is the larger of *maxagents*/50 x *max_querydegree* or *maxagents* - *max_coordagents*. Using the default, the value for an database partition server is the larger of *maxagents*/10 x *max_querydegree* or *maxagents* - *max_coordagents*.
Related Parameters	

- “Initial Number of Agents in Pool (num_initagents)”
- “Maximum Number of Agents (maxagents)” on page 379
- “Maximum Query Degree of Parallelism (max_querydegree)” on page 443
- “Maximum Number of Coordinating Agents (max_coordagents)” on page 381

This parameter is a guideline for how large you want the agent pool to grow (and replaces the *max_idleagents* parameter that was used in DB2 Version 2).

The agent pool contains subagents and idle agents. Idle agents can be used as parallel subagents or as coordinating agents. If more agents are created than is indicated by the value of this parameter, they will be terminated when they finish executing their current request, rather than be returned to the pool.

If the value for this parameter is 0, agents will be created as needed, and may be terminated when they finish executing their current request. If the value is *maxagents*, and the pool is full of associated subagents, the server cannot be used as a coordinator node, because no new coordinating agents can be created.

Recommendation: If you run a decision-support environment in which few applications connect concurrently, set *num_poolagents* to a small value to avoid having an agent pool that is full of idle agents.

If you run a transaction-processing environment in which many applications are concurrently connected, increase the value of *num_poolagents* to avoid the costs associated with the frequent creation and termination of agents.

Initial Number of Agents in Pool (num_initagents)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	0 [0 — *num_poolagents*]
Related Parameters	

- "Maximum Number of Agents (maxagents)" on page 379
- "Agent Pool Size (num_poolagents)" on page 383
- "Maximum Number of Coordinating Agents (max_coordagents)" on page 381

This parameter determines the initial number of idle agents that are created in the agent pool at DB2START time.

Database Application Remote Interface (DARI)

The following parameters can affect the Database Application Remote Interface (DARI) applications:

- "Keep DARI Process Indicator (keepdari)"
- "Maximum Number of DARI Processes (maxdari)" on page 386
- "Initialize DARI Process with JVM (initdari_jvm)" on page 387
- "Initial Number of Fenced DARI Processes in Pool (num_initdaris)" on page 388

Note: The term DARI refers to stored procedures.

Keep DARI Process Indicator (keepdari)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	Yes [Yes; No]
Related Parameters	"Maximum Number of DARI Processes (maxdari)" on page 386

This parameter indicates whether or not a DARI process is kept after a DARI call is complete. DARI processes are created as separate system entities in order to isolate user-written DARI code from the database manager agent process. This parameter is only applicable on database servers.

If *keepdari* is set to *no*, a new DARI process is created and destroyed for each DARI invocation. If *keepdari* is set to *yes*, a DARI process is reused for subsequent DARI calls. When the database manager is stopped, all outstanding DARI processes will be terminated.

Setting this parameter to *yes* will result in additional system resources being consumed by the database manager for each DARI process that is activated, up to the value contained in the *maxdari* parameter. This is only true when no existing DARI process is available to process a subsequent DARI call. This parameter is ignored if *maxdari* is set to 0.

Recommendation: In an environment in which the number of DARI requests is large relative to the number of non-DARI requests, and system resources are not constrained, then this parameter can be set to *yes*. This will improve the DARI performance by avoiding the initial DARI process creation overhead since an existing DARI process will be used to process the call.

For example, in an OLTP debit-credit banking transaction application, the code to perform each transaction could be performed in a stored procedure which executes in a DARI process. In this application, the main workload is performed out of DARI processes. If this parameter is set to *no*, each transaction incurs the overhead of creating a new DARI process, resulting in a significant performance reduction. If, however, this parameter is set to *yes*, each transaction would try to use an existing DARI process, which would avoid this overhead.

Maximum Number of DARI Processes (maxdari)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	-1 (*max_coordagents*) [-1; 0 – *max_coordagents*]
Unit of Measure	Counter
Related Parameters	• "Maximum Number of Agents (maxagents)" on page 379 • "Keep DARI Process Indicator (keepdari)" on page 385 • "Initial Number of Fenced DARI Processes in Pool (num_initdaris)" on page 388

- “Maximum Number of Coordinating Agents (max_coordagents)” on page 381

This parameter indicates the maximum number of DARI process that may reside at the database server. Once this limit is reached, no new DARIrequests may be invoked. This parameter is only applicable on database servers.

There can be no more than one DARI process active per coordinating agent, so the maximum number of DARI processes is also dictated by the maximum number of coordinating agents (*max_coordagents*).

Recommendation: If your environment features the use of the DARI facility within the database manager, then this parameter can be used to ensure that an appropriate number of DARI processes are available to handle the DARI calls made at any one time within the database manager.

If the parameter is set to −1, the maximum number of DARIprocesses will be the same as the value set in the *max_coordagents* parameter.

If you find that the default value is not appropriate for your environment because an inappropriate amount of system resource is being given to DARI processes which is affecting performance of the database manager, the following may be useful in providing a starting point for tuning this parameter:

```
   maxdari = # of applications allowed to make DARI calls at one time
```

If *keepdari* is set to *yes*, then each DARIprocess that is created will continue to exist and use system resources even after the DARI call has been processed and returned to the agent.

If your environment is tightly constrained and you cannot afford the process resources associated with DARI, you can disable DARIby setting this parameter to zero (0).

Initialize DARI Process with JVM (initdari_jvm)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable

Default [Range]	No [Yes; No]
Related Parameters	• "Maximum Number of DARI Processes (maxdari)" on page 386 • "Initial Number of Fenced DARI Processes in Pool (num_initdaris)" • "Keep DARI Process Indicator (keepdari)" on page 385

This parameter indicates whether each fenced DARI process will load the Java Virtual Machine (JVM) when starting. This parameter will reduce the initial startup time for fenced Java stored procedures, especially when used in conjunction with the *num_initdaris* parameter. This parameter could increase the initial load time for non-Java fenced stored procedures as they do not require the JVM.

Initial Number of Fenced DARI Processes in Pool (num_initdaris)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	0 [0 — *maxdari*]
Related Parameters	• "Maximum Number of DARI Processes (maxdari)" on page 386 • "Initialize DARI Process with JVM (initdari_jvm)" on page 387 • "Keep DARI Process Indicator (keepdari)" on page 385

This parameter indicates the initial number of idle fenced DARI processes that are created in the DARI pool at DB2START time. Setting this parameter will reduce the initial startup time for fenced stored procedures. This parameter is ignored if *keepdari* is not specified.

Logging and Recovery

Recovering your environment can be very important to prevent the loss of critical data. A number of parameters are available to help you manage your environment and to ensure that you can perform adequate recovery of your data or transactions. These parameters are grouped into the following categories:

- "Database Log Files"
- "Database Log Activity" on page 395
- "Recovery" on page 400
- "Distributed Unit of Work Recovery" on page 406.

Database Log Files

The following parameters provide information about number, size and status of the files used for database logging:

- "Size of Log Files (logfilsiz)"
- "Number of Primary Log Files (logprimary)" on page 391
- "Number of Secondary Log Files (logsecond)" on page 392
- "Change the Database Log Path (newlogpath)" on page 393
- "Location of Log Files (logpath)" on page 395
- "First Active Log File (loghead)" on page 395

Size of Log Files (logfilsiz)

Configuration Type	Database	
Parameter Type	Configurable	
Default [Range]		
	UNIX	1000 [4 — 65 535]
	Windows NT	250 [4 — 65 535]
	OS/2	250 [4 — 65 535]
Unit of Measure	Pages (4 KB)	

Related Parameters

- "Number of Primary Log Files (logprimary)" on page 391
- "Number of Secondary Log Files (logsecond)" on page 392
- "Recovery Range and Soft Checkpoint Interval (softmax)" on page 397

This parameter defines the size of each primary and secondary log file. The size of these log files limits the number of log records that can be written to them before they become full and a new log file is required.

The use of primary and secondary log files as well as the action taken when a log file becomes full are dependent on the type of logging that is being performed:

- Circular logging

  A primary log file can be reused when the changes recorded in it have been committed. If the log file size is small and applications have processed a large number of changes to the database without committing the changes, a primary log file can quickly become full. If all primary log files become full, the database manager will allocate secondary log files to hold the new log records.
- Log Retention logging

  When a primary log file is full, the log is archived and a new primary log file is allocated.

Recommendation: You must balance the size of the log files with the number of primary log files:

- The value of the *logfilsiz* should be increased if the database has a large number of update, delete and/or insert transactions running against it which will cause the log file to become full very quickly.

  Note: The total log file size limit is 32 GB. That is, the number of log files (*logprimary* + *logsecond*) multiplied by the size of each log file in bytes (*logfilsiz* * 4096) must be less than 32 GB.

  A log file that is too small can affect system performance because of the overhead of archiving old log files, allocating new log files, and waiting for a usable log file.
- The value of the *logfilsiz* should be reduced if disk space is scarce, since primary logs are preallocated at this size.

  A log file that is too large can reduce your flexibility when managing archived log files and copies of log files, since some media may not be able to hold an entire log file.

If you are using log retention, the current active log file is closed and truncated when the last application disconnects from a database. When the next connection to the database occurs, the next log file is used. Therefore, if you understand the logging requirements of your concurrent applications you may be able to determine a log file size which will not allocate excessive amounts of wasted space.

Refer to “Configuration Parameters for Database Logging” in the *Administration Guide: Implementation* for more information on this parameter.

Number of Primary Log Files (logprimary)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	3 [2 – 128]
Unit of Measure	Counter

When Allocated

- The database is created
- A log is moved to a different location (which occurs when the *logpath* parameter is updated)
- Following a increase in the value of this parameter (*logprimary*), during the next database connection after all users have disconnected
- A log file has been archived and a new log file is allocated (the *logretain* or *userexit* parameter must be enabled)
- If the *logfilsiz* parameter has been changed, the active log files are re-sized during the next database connection after all users have disconnected.

When Freed
Not freed unless this parameter decreases. If decreased, unneeded log files are deleted during the next connection to the database.

Related Parameters

- “Size of Log Files (logfilsiz)” on page 389
- “Number of Secondary Log Files (logsecond)” on page 392
- “Log Retain Enable (logretain)” on page 399
- “User Exit Enable (userexit)” on page 399

The primary log files establish a fixed amount of storage allocated to the recovery log files. This parameter allows you to specify the number of primary log files to be preallocated.

Under circular logging, the primary logs are used repeatedly in sequence. That is, when a log is full, the next primary log in the sequence is used if it is available. A log is considered available if all units of work with log records in it have been committed or rolled-back. If the next primary log in sequence is not available, then a secondary log is allocated and used. Additional secondary logs are allocated and used until the next primary log in the sequence becomes available or the limit imposed by the *logsecond* parameter is reached. These secondary log files are dynamically deallocated as they are no longer needed by the database manager.

The number of primary and secondary log files must comply with the following equation:
- (*logprimary* + *logsecond*) <= 128

Recommendation: The value chosen for this parameter depends on a number of factors, including the type of logging being used, the size of the log files, and the type of processing environment (for example, length of transactions and frequency of commits).

Increasing this value will increase the disk requirements for the logs because the primary log files are preallocated during the very first connection to the database.

If you find that secondary log files are frequently being allocated, you may be able to improve system performance by increasing the log file size (*logfilsiz*) or by increasing the number of primary log files.

For databases that are not frequently accessed, in order to save disk storage, set the parameter to 2. For databases enabled for roll-forward recovery, set the parameter larger to avoid the overhead of allocating new logs almost immediately.

You may use the database system monitor to help you size the primary log files.

For more information see the following monitor element descriptions in the *System Monitor Guide and Reference*:
- *sec_log_used_top (maximum secondary log space used)*
- *tot_log_used_top (maximum total log space used)*
- *sec_logs_allocated (secondary logs allocated currently)*

Observation of these monitor values over a period of time will aid in better tuning decisions, as average values may be more representative of your ongoing requirements.

Number of Secondary Log Files (logsecond)

Configuration Type	Database
Parameter Type	Configurable

Default [Range]	2 [0 – 126]
Unit of Measure	Counter
When Allocated	As needed when *logprimary* is insufficient (see detail below)
When Freed	Over time as the database manager determines they will no longer be required.
Related Parameters	• "Size of Log Files (logfilsiz)" on page 389 • "Number of Primary Log Files (logprimary)" on page 391 • "Log Retain Enable (logretain)" on page 399 • "User Exit Enable (userexit)" on page 399

This parameter specifies the number of secondary log files that are created and used for recovery log files (only as needed). When the primary log files become full, the secondary log files (of size *logfilsiz*) are allocated one at a time as needed, up to a maximum number as controlled by this parameter. An error code will be returned to the application, and the database will be shutdown, if more secondary log files are required than are allowed by this parameter.

See "Number of Primary Log Files (logprimary)" on page 391 for more information about how secondary logs are used.

Recommendation: Use secondary log files for databases that have periodic needs for large amounts of log space. For example, an application that is run once a month may require log space beyond that provided by the primary log files. Since secondary log files do not require permanent file space they are advantageous in this type of situation.

Change the Database Log Path (newlogpath)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any valid path or device]
Related Parameters	• "Location of Log Files (logpath)" on page 395 • "Database is Consistent (database_consistent)" on page 417

This parameter allows you to specify a string of up to 242 bytes to change the location where the log files are stored. The string can point to either a path name, or to a raw device. If the string points to a path name, it must be a fully qualified path name, not a relative path name.

Note: In a partitioned database environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

To specify a device, specify a string that the operating system identifies as a device. For example:

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

  Note: You must have Windows NT Version 4.0 with Service Pack 3 installed to be able to write logs to a device.
- On UNIX-based platforms, `/dev/rdblog8`

Note: You can only specify a device on AIX, Windows NT, and Solaris platforms.

The new setting does not become the value of *logpath* until both of the following occur:

- The database is in a consistent state, as indicated by the *database_consistent* parameter.
- All users are disconnected from the database

When the first new connection is made to the database, the database manager will move the logs to the new location specified by *logpath*.

There might be log files in the old log path. These log files might not have been archived. You might need to archive these log files manually. Also, if you are running replication on this database, replication might still need the log files from before the log path change. If the database is configured with the User Exit Enable (*userexit*) database configuration parameter set to "Yes", and if all the log files have been archived either by DB2 automatically or by yourself manually, then DB2 will be able to retrieve the log files to complete the replication process. Otherwise, you can copy the files from the old log path to the new log path.

Recommendation: Ideally, the log files will be on a physical disk which does **not** have high I/O. For instance, avoid putting the logs on the same disk as the operating system or high volume databases. This will allow for efficient logging activity with a minimum of overhead such as waiting for I/O.

You may use the database system monitor to track the number of I/O's related to database logging.

For more information, refer to the following monitor element descriptions in the *System Monitor Guide and Reference*:
- *log_reads* (number of log pages read)
- *log_writes* (number of log pages written).

The preceding data elements return the amount of I/O activity related to database logging. You can use an operating system monitor tool to collect information about other disk I/O activity, then compare the two types of I/O activity.

Location of Log Files (logpath)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	"Change the Database Log Path (newlogpath)" on page 393

This parameter contains the current path being used for logging purposes. You cannot change this parameter directly as it is set by the database manager after a change to the *newlogpath* parameter becomes effective.

When a database is created, the recovery log file for it is created in a subdirectory of the directory containing the database. The default is a subdirectory named SQLOGDIR under the directory created for the database.

First Active Log File (loghead)

Configuration Type	Database
Parameter Type	Informational

This parameter contains the name of the log file that is currently active.

Database Log Activity

The following parameters can influence the type and performance of database logging:
- "Number of Commits to Group (mincommit)"
- "Recovery Range and Soft Checkpoint Interval (softmax)" on page 397
- "Log Retain Enable (logretain)" on page 399
- "User Exit Enable (userexit)" on page 399

Number of Commits to Group (mincommit)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	1 [1 – 25]

Unit of Measure	Counter

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed. This delay can help reduce the database manager overhead associated with writing log records and as a result improve performance when you have multiple applications running against a database and many commits are requested by the applications within a very short time frame.

This grouping of commits will only occur when the value of this parameter is greater than one and when the number of applications connected to the database is greater than or equal to the value of this parameter. When commit grouping is being performed, application commit requests are held until either one second has elapsed or the number of commit requests equals the value of this parameter.

Changes to the value specified for this parameter take effect immediately; you do not have to wait until all applications disconnect from the database.

Recommendation: Increase this parameter from its default value if multiple read/write applications typically request concurrent database commits. This will result in more efficient logging file I/O as it will occur less frequently and write more log records each time it does occur.

You could also sample the number of transactions per second and adjust this parameter to accommodate the peak number of transactions per second (or some large percentage of it). Accommodating peak activity would minimize the overhead of writing log records during heavy load periods.

If you increase *mincommit*, you may also need to increase the *logbufsz* parameter to avoid having a full log buffer force a write during these heavy load periods. In this case, the *logbufsz* should be equal to:

```
mincommit * (log space used, on average, by a transaction)
```

You may use the database system monitor to help you tune this parameter in the following ways:
- Calculating the peak number of transactions per second:

  Taking monitor samples throughout a typical day, you can determine your heavy load periods. You can calculate the total transactions by adding the following monitor elements:
  - *commit_sql_stmts (commit statements attempted)*
  - *rollback_sql_stmts (rollback statements attempted)*

  Using this information and the available timestamps, you can calculate the number of transactions per second.
- Calculating the log space used per transaction:

Using sampling techniques over a period of time and a number of transactions, you can calculate an average of the log space used with the following monitor element:
– *log_space_used (unit of work log space used)*

For more information about the database system monitor, see the *System Monitor Guide and Reference*.

Recovery Range and Soft Checkpoint Interval (softmax)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	100 [1 – 100 * *logprimary*]
Unit of Measure	Percentage of total number of primary log files
Related Parameters	• "Size of Log Files (logfilsiz)" on page 389 • "Number of Primary Log Files (logprimary)" on page 391

This parameter is used to:
- Influence the number of logs that need to be recovered following a crash (such as a power failure). For example, if the default value is used, the database manager will try to keep the number of logs that need to be recovered to 1. If you specify 300 as the value of this parameter, the database manager will try to keep the number of logs that need to be recovered to 3.

  To influence the number of logs required for crash recovery, the database manager uses this parameter to trigger the page cleaners to ensure that pages older than the specified recovery window are already written to disk.
- Determine the frequency of soft checkpoints.

At the time of a database failure resulting from an event such as a power failure, there may have been changes to the database which:
- Have not been committed, but updated the data in the buffer pool
- Have been committed, but have not been written from the buffer pool to the disk
- Have been committed and written from the buffer pool to the disk.

When a database is restarted, the log files will be used to perform a crash recovery of the database which ensures that the database is left in a consistent state (that is, all committed transactions are applied to the database and all uncommitted transactions are not applied to the database).

To determine which records from the log file need to be applied to the database, the database manager uses a log control file. This log control file is periodically written to disk, and, depending on the frequency of this event, the database manager may be applying log records of committed transactions or applying log records that describe changes that have already been written from the buffer pool to disk. These log records have no impact on the database, but applying them introduces some overhead into the database restart process.

The log control file is always written to disk when a log file is full, and during soft checkpoints. You can use this configuration parameter to trigger additional soft checkpoints.

The timing of soft checkpoints is based on the difference between the “current state” and the “recorded state”, given as a percentage of the *logfilsiz*. The “recorded state” is determined by the oldest valid log record indicated in the log control file on disk, while the “current state” is determined by the log control information in memory. (The oldest valid log record is the first log record that the recovery process would read.) The soft checkpoint will be taken if the value calculated by the following formula is greater than or equal to the value of this parameter:

```
  ( (space between recorded and current states) / logfilsiz ) * 100 * logprimary
```

Recommendation: You may want to increase or reduce the value of this parameter, depending on whether your acceptable recovery window is greater than or less than one log file. Lowering the value of this parameter will cause the database manager both to trigger the page cleaners more often and to take more frequent soft checkpoints. These actions can reduce both the number of log records that need to be processed and the number of redundant log records that are processed during crash recovery.

Note however, that more page cleaner triggers and more frequent soft checkpoints increase the overhead associated with database logging, which can impact the performance of the database manager. Also, more frequent soft checkpoints may not reduce the time required to restart a database, if you have:
- Very long transactions with few commit points.
- A very large buffer pool and the pages containing the committed transactions are not written back to disk very frequently. (Note that the use of asynchronous page cleaners can help avoid this situation. See “Number of Asynchronous Page Cleaners (num_iocleaners)” on page 367.)

In both of these cases, the log control information kept in memory does not change frequently and there is no advantage in writing the log control information to disk, unless it has changed.

Log Retain Enable (logretain)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Recovery; Capture; No]
Related Parameters	• "User Exit Enable (userexit)" • "Log Retain Status Indicator (log_retain_status)" on page 418 • "Backup Pending Indicator (backup_pending)" on page 417

The values are as follows:

- No, to indicate that logs are not retained.
- Recovery, to indicate that the logs are retained, and can be used for forward recovery. In addition, if you are using data replication, the Capture program can write the updates recorded in the logs to the change table.
- Capture, to indicate that the logs are only retained so that the Capture program can write the updates to the change table. These logs may be used for forward recovery if they have not been pruned.

If *logretain* is set to "Recovery" or *userexit* is set to "Yes", the active log files will be retained and become online archive log files for use in roll-forward recovery. This is called log retention logging.

After *logretain* is set to "Recovery" or *userexit* is set to "Yes" (or both), you must make a full backup of the database. This state is indicated by the *backup_pending* flag parameter.

If *logretain* is set to "No" and *userexit* is set to "No", roll-forward recovery is not available for the database.

When *logretain* is set to "Capture", the Capture program calls the PRUNE LOGFILE command to delete log files when the Capture program completes. You should not set *logretain* to "Capture" if you want to perform roll-forward recovery on the database.

If *logretain* is set to "No" and *userexit* is set to "No", logs are not retained. In this situation, the database manager deletes all log files in the *logpath* directory (including online archive log files), allocates new active log files, and reverts to circular logging.

User Exit Enable (userexit)

Configuration Type	Database

Parameter Type	Configurable
Default [Range]	No [Yes; No]
Related Parameters	• “Log Retain Enable (logretain)” on page 399 • “User Exit Status Indicator (user_exit_status)” on page 418 • “Backup Pending Indicator (backup_pending)” on page 417

If this parameter is enabled, log retention logging is performed regardless of how the *logretain* parameter is set. This parameter also indicates that a user exit program should be used to archive and retrieve the log files. Log files are archived when the database manager closes the log file. They are retrieved when the ROLLFORWARD utility needs to use them to restore a database.

After *logretain*, or *userexit*, or both of these parameters are enabled, you must make a full backup of the database. This state is indicated by the *backup_pending* flag parameter.

If both of these parameters are de-selected, roll-forward recovery becomes unavailable for the database because logs will no longer be retained. In this case, the database manager deletes all log files in the *logpath* directory (including online archive log files), allocates new active log files, and reverts to circular logging.

Refer to “User Exit for Database Recovery” in the *Administration Guide: Implementation* for more information on the user exit program.

Recovery

The following parameters affect various aspects of database recovery:

- “Auto Restart Enable (autorestart)” on page 401
- “Index Re-creation Time (indexrec)” on page 401
- “Default Number of Load Recovery Sessions (dft_loadrec_ses)” on page 402
- “Recovery History Retention Period (rec_his_retentn)” on page 404
- “Number of Database Backups (num_db_backups)” on page 403

See also “Distributed Unit of Work Recovery” on page 406.

The following parameters are used when working with Tivoli Storage Manager (TSM):

- “Tivoli Storage Manager Management Class (tsm_mgmtclass)” on page 404
- “Tivoli Storage Manager Password (tsm_password)” on page 405
- “Tivoli Storage Manager Node Name (tsm_nodename)” on page 405

- “Tivoli Storage Manager Owner Name (tsm_owner)” on page 406

Auto Restart Enable (autorestart)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	On [On; Off]

When this parameter is set on, the database manager automatically calls the restart database utility, if needed, when an application connects to a database. *Crash recovery* is the operation performed by the restart database utility. It is performed if the database terminated abnormally while applications were connected to it. An abnormal termination of the database could be caused by a power failure or a system software failure. It applies any committed transactions that were in the database buffer pool but were not written to disk at the time of the failure. It also backs out any uncommitted transactions that may have been written to disk.

If *autorestart* is not enabled, then an application that attempts to connect to a database which needs to have crash recovery performed (needs to be restarted) will receive a SQL1015N error. In this case, the application can call the restart database utility, or you can restart the database by selecting the restart operation of the recovery tool.

Index Re-creation Time (indexrec)

Configuration Type	Database and Database Manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	**UNIX Database Manager** restart [restart; access] **OS/2 and Windows NT Database Manager** access [restart; access] **Database** Use system setting [system; restart; access]
Related Parameters	“Auto Restart Enable (autorestart)”

This parameter indicates when the database manager will attempt to rebuild invalid indexes. There are three possible settings for this parameter:

SYSTEM	*use system setting* which will cause invalid indexes to be rebuilt at the time specified in the database manager configuration file. (Note: This setting is only valid for database configurations.)
ACCESS	*during index access* which will cause invalid indexes to be rebuilt when the index is first accessed.
RESTART	*during database restart* which will cause invalid indexes to be rebuilt when a RESTART DATABASE command is either explicitly or implicitly issued. Note that a RESTART DATABASE command is implicitly issued if the *autorestart* parameter is enabled.

For the numeric equivalents and API constants for these values, refer to the *Administrative API Reference*.

Indexes can become invalid when fatal disk problems occur. If this happens to the data itself, the data could be lost. However, if this happens to an index, the index can be recovered by re-creating it. If an index is rebuilt while users are connected to the database, two problems could occur:

- An unexpected degradation in response time may occur as the index file is re-created. Users accessing the table and using this particular index would wait while the index was being rebuilt.
- Unexpected locks may be held after index re-creation, especially if the user transaction that caused the index to be re-created never performed a COMMIT or ROLLBACK.

Recommendation: The best choice for this option on a high-user server and if restart time is not a concern, would be to have the index rebuilt at DATABASE RESTART time as part of the process of bringing the database back online after a crash.

Setting this parameter to “ACCESS” will result in a degradation of the performance of the database manager while the index is being re-created. Any user accessing that specific index or table would have to wait until the re-creating is complete.

If this parameter is set to “RESTART”, the time taken to restart the database will be longer due to index re-creation but normal processing would not be impacted once the database has been brought back online.

Default Number of Load Recovery Sessions (dft_loadrec_ses)

Configuration Type	Database

Parameter Type	Configurable
Default [Range]	1 [1 – 30 000]
Unit of Measurement	Counter

This parameter specifies the default number of sessions that will be used during the recovery of a table load. The value should be set to an optimal number of I/O sessions to be used to retrieve a load copy. The retrieval of a load copy is an operation similar to restore. You can override this parameter through entries in the copy location file specified by the environment variable DB2LOADREC.

The default number of buffers used for load retrieval is two more than the value of this parameter. You can also override the number of buffers in the copy location file.

This parameter is applicable only if roll forward recovery is enabled.

Refer to *Data Movement Utilities Guide and Reference* for more information about load recovery.

Number of Database Backups (num_db_backups)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	12 [1 — 32 768]
Related Parameters	"Recovery History Retention Period (rec_his_retentn)" on page 404

This parameter specifies the number of database backups to retain for a database. After the specified number of backups is reached, old backups are marked as expired in the recovery history file. Recovery history file entries for the table space backups and load copy backups that are related to the expired database backup are also marked as expired. When a backup is marked as expired, the physical backups can be removed from where they are stored (for example, disk, tape, ADSM). The next database backup will prune the expired entries from the recovery history file.

When a database backup is marked as expired in the history file, any corresponding file backups linked through a DB2 Data Links Manager will be removed from its archive server.

The *rec_his_retentn* configuration parameter should be set to a value compatible with the value of *num_db_backups*. For example, if *num_db_backup* is set to a large value, the value for *rec_his_retentn* should be large enough to support that number of backups.

Recovery History Retention Period (rec_his_retentn)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	366 [-1; 0 — 30 000]
Unit of Measure	Days
Related Parameters	"Number of Database Backups (num_db_backups)" on page 403

This parameter is used to specify the number of days that historical information on backups should be retained. If the recovery history file is not needed to keep track of backups, restores, and loads, this parameter can be set to a small number.

If value of this parameter is -1, the recovery history file can only be pruned explicitly using the available commands or APIs. If the value is not -1, the recovery history file is pruned after every full database backup.

The value of this parameter will override the value of the *num_db_backups* parameter, but *rec_his_retentn* and *num_db_backups* must work together. If the value for *num_db_backups* is large, the value for *rec_his_retentn* should be large enough to support that number of backups.

No matter how small the retention period, the most recent full database backup plus its restore set will always be kept, unless you use the PRUNE utility with the FORCE option. For more information about this utility, refer to the *Command Reference*.

Tivoli Storage Manager Management Class (tsm_mgmtclass)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any string]

The Tivoli Storage Manager management class tells how the TSM server should manage the backup versions of the objects being backed up.

The default is that there is no TSM management class.

The management class is assigned from the Tivoli Storage Manager administrator. Once assigned, you should set this parameter to the management class name. When performing any TSM backup, the database manager uses this parameter to pass the management class to TSM.

Refer to "Tivoli Storage Manager" in the *Administration Guide: Implementation* for more information on Tivoli Storage Manager.

Tivoli Storage Manager Password (tsm_password)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any string]

This parameter is used to override the default setting for the password associated with the Tivoli Storage Manager (TSM) product. The password is needed to allow you to restore a database that was backed up to TSM from another node.

Note: If the *tsm_nodename* is overridden during a backup done with DB2 (for example, with the BACKUP DATABASE command), the *tsm_password* may also have to be set.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the *tsm_nodename* to be overridden during a backup done with DB2.

Refer to "Tivoli Storage Manager" in the *Administration Guide: Implementation* for more information on Tivoli Storage Manager.

Tivoli Storage Manager Node Name (tsm_nodename)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any string]

This parameter is used to override the default setting for the node name associated with the Tivoli Storage Manager (TSM) product. The node name is needed to allow you to restore a database that was backed up to TSM from another node.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the *tsm_nodename* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

Refer to "Tivoli Storage Manager" in the *Administration Guide: Implementation* for more information on Tivoli Storage Manager.

Tivoli Storage Manager Owner Name (tsm_owner)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any string]

This parameter is used to override the default setting for the owner associated with the Tivoli Storage Manager (TSM) product. The owner name is needed to allow you to restore a database that was backed up to ADSM from another node. It is possible for the *tsm_owner* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

Note: The owner name is case sensitive.

The default is that you can only restore a database from TSM on the same node from which you did the backup.

Refer to "Tivoli Storage Manager" in the *Administration Guide: Implementation* for more information on Tivoli Storage Manager.

Distributed Unit of Work Recovery

The following parameters affect the recovery of Distributed Unit of Work (DUOW) transactions:

- "Transaction Manager Database Name (tm_database)"
- "Transaction Resync Interval (resync_interval)" on page 407
- "Sync Point Manager Log File Path (spm_log_path)" on page 408
- "Sync Point Manager Name (spm_name)" on page 408
- "Sync Point Manager Log File Size (spm_log_file_sz)" on page 409
- "Sync Point Manager Resync Agent Limit (spm_max_resync)" on page 410

Transaction Manager Database Name (tm_database)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients

	• Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	1ST_CONN [any valid database name]

This parameter identifies the name of the Transaction Manager (TM) database for each DB2 instance. A TM database can be:

- A local DB2 Universal Database database
- A remote DB2 Universal Database database that does not reside on a host or AS/400 system
- A DB2 for OS/390 Version 5 database if accessed via TCP/IP and the Sync Point Manager is not used.

The TM database is a database that is used as a logger and coordinator, and is used to perform recovery for indoubt transactions.

You may set this parameter to **1ST_CONN** which will set the TM database to be the first database to which a user connects.

Refer to "Distributed Databases" in the *Administration Guide: Planning* for more information on distributed unit of work.

Recommendation: For simplified administration and operation you may wish to create a few databases over a number of instances and use these databases exclusively as TM databases.

Transaction Resync Interval (resync_interval)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	180 [1 – 60 000]
Unit of Measurement	Seconds

This parameter specifies the time interval in seconds for which a Transaction Manager (TM), Resource Manager (RM) or Sync Point Manager (SPM) should retry the recovery of any outstanding indoubt transactions found in the TM, the RM, or the SPM. This parameter is applicable when you have transactions running in a distributed unit of work (DUOW) environment.

Refer to "Distributed Databases" in the *Administration Guide: Planning* for more information on distributed unit of work.

Recommendation: If, in your environment, indoubt transactions will not interfere with other transactions against your database, you may wish to increase the value of this parameter. If you are using a DB2 Connect gateway to access DRDA2 Application Servers, you should consider the effect indoubt transactions may have at the Application Servers even though there will be no interference with local data access. If there are no indoubt transactions, the performance impact will be minimal.

Sync Point Manager Log File Path (spm_log_path)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default	sqllib/spmlog [any valid path or device]

This parameter specifies the directory where the Sync Point Manager (SPM) logs are written. By default, the logs are written to the `sqllib/spmlog` directory, which, in a high-volume transaction environment, can cause an I/O bottleneck. Use this parameter to have the SPM log files placed on a faster disk than the current `sqllib/spmlog` directory. This allows for better concurrency among the SPM agents.

For more information on the Sync Point Manager, refer to the *Installation and Configuration Supplement*.

Refer to "Recovery of Indoubt Transactions on the Host" in the *Administration Guide: Planning* for more information on recovery of indoubt DRDA transactions.

Sync Point Manager Name (spm_name)

Configuration Type	Database manager
Applies to	

	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default	Derived from the TCP/IP hostname

This parameter identifies the name of the Sync Point Manager (SPM) instance to the database manager.

For more information on the Sync Point Manager, refer to the *Installation and Configuration Supplement*.

Refer to “Recovery of Indoubt Transactions on the Host” in the *Administration Guide: Planning* for more information on recovery of indoubt DRDA transactions.

Sync Point Manager Log File Size (spm_log_file_sz)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	256 [4 — 1 000]
Unit of Measure	Pages

This parameter identifies the Sync Point Manager (SPM) log file size in 4 KB pages. The log file is contained in the `spmlog` sub-directory under `sqllib` and is created the first time SPM is started.

For more information on the Sync Point Manager, refer to the *Installation and Configuration Supplement*.

Refer to “Recovery of Indoubt Transactions on the Host” in the *Administration Guide: Planning* for more information on recovery of indoubt DRDA transactions.

Recommendation: The Sync Point Manager log file size should be large enough to maintain performance, but small enough to prevent wasted space. The size required depends on the number of transactions using protected conversations, and how often COMMIT or ROLLBACK is issued.

To change the size of the SPM log file:
1. Determine that there are no indoubt transactions by using the LIST DRDA INDOUBT TRANSACTIONS command.
2. If there are none, stop the database manager.
3. Update the Database Manager Configuration with a new SPM log file size.
4. Go to the `$HOME/sqllib` directory and issue `rm -fr spmlog` to delete the current SPM log. (Note: This shows the AIX command. Other systems may require a different remove or delete command.)
5. Start the database manager. (A new SPM log of the specified size is created during the startup of the database manager.)

Sync Point Manager Resync Agent Limit (spm_max_resync)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	20 [10 — 256]

This parameter identifies the number of agents that can simultaneously perform resync operations.

Refer to “Recovery of Indoubt Transactions on the Host” in the *Administration Guide: Planning* for more information on recovery of indoubt DRDA transactions.

For more information on the Sync Point Manager, refer to the *Installation and Configuration Supplement*.

Database Management

A number of parameters are available which provide information about your database or influence the management of your database. These are grouped as follows:
- “Query Enabler” on page 411

- “Attributes”
- “DB2 Data Links Manager” on page 414
- “Status” on page 417
- “Compiler Settings” on page 419.

Query Enabler

The following parameters provide information for the control of Query Enabler:

- “Dynamic SQL Query Management (dyn_query_mgmt)”

Dynamic SQL Query Management (dyn_query_mgmt)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	0 (DISABLE) [1(ENABLE), 0 (DISABLE)]

This parameter is relevant where DB2 Query Patroller is installed. If the database configuration parameter *dyn_query_mgmt* is set to “ENABLE” and the cost of the dynamic query exceeds the trap_threshold for the user or group (as specified in the DB2 Query Patroller user profile table), then this query will be caught by DB2 Query Patroller. The trap_threshold is a cost-based trigger for query catching established in DB2 Query Patroller by the user. When a dynamic query is caught, a dialog will be presented for the user to specify runtime parameters.

If *dyn_query_mgmt* is set to “DISABLE”, then no queries will be caught.

Attributes

The following parameters provide general information about the database:

- “Configuration File Release Level (release)”
- “Database Release Level (database_level)” on page 412
- “Territory for the Database (territory)” on page 412
- “Country code for the Database (country)” on page 412
- “Codeset for the Database (codeset)” on page 413
- “Code Page for the Database (codepage)” on page 413
- “Collating Information (collate_info)” on page 413
- “Copy Protection Enable (copyprotect)” on page 414

With the exception of *copyprotect*, these parameters are provided for informational purposes only.

Configuration File Release Level (release)

Configuration Type	Database manager, Database

Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Informational
Related Parameters	"Database Release Level (database_level)"

This parameter specifies the release level of the configuration file.

Database Release Level (database_level)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	"Configuration File Release Level (release)" on page 411

This parameter indicates the release level of the database manager which can use the database. In the case of an incomplete or failed migration, this parameter will reflect the release level of the unmigrated database and may differ from the *release* parameter (the release level of the database configuration file). Otherwise the value of *database_level* will be identical to value of the *release* parameter.

Territory for the Database (territory)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	"Country code for the Database (country)"

This parameter shows the territory used to create the database. Territory is used by the database manager to determine *country* parameter values. For more information about how the database manager uses the territory, see the *Quick Beginnings* .

Country code for the Database (country)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	"Territory for the Database (territory)"

This parameter shows the country code used to create the database. The *country* parameter is derived based on the *territory* parameter. For more information, see the *Quick Beginnings* .

Codeset for the Database (codeset)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	"Code Page for the Database (codepage)"

This parameter shows the codeset that was used to create the database. Codeset is used by the database manager to determine *codepage* parameter values. For more information about how the database manager uses the codeset, see the *Quick Beginnings* .

Code Page for the Database (codepage)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	"Codeset for the Database (codeset)"

This parameter shows the code page that was used to create the database. The *codepage* parameter is derived based on the *codeset* parameter. For more information, see the *Quick Beginnings* .

Collating Information (collate_info)

This parameter can only be displayed using the GET DATABASE CONFIGURATION API. It **cannot** be displayed through the command line processor or the Control Center.

Configuration Type	Database
Parameter Type	Informational

This parameter provides 260 bytes of database collating information. The first 256 bytes specify the database collating sequence, where byte "n" contains the sort weight of the code point whose underlying decimal representation is "n" in the code page of the database.

The last 4 bytes contain internal information about the type of the collating sequence. You can treat it as an integer applicable to the platform of the database. There are three values:
- **0 –** The sequence contains non-unique weights
- **1 –** The sequence contains all unique weights

- **2 –** The sequence is the identity sequence, for which strings are compared byte for byte.

If you use this internal type information, you need to consider byte reversal when retrieving information for a database on a different platform.

You can specify the collating sequence at database creation time.

Copy Protection Enable (copyprotect)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Yes; No]

This parameter enables the copy-protect attribute and is disabled by default. Prior to Version 2 of the database manager, the default was to enable the copy-protect attribute.

This parameter does not apply to UNIX-based environments.

The backup database and restore database utilities are not affected by the *copyprotect* parameter. It is possible to back up a copy-protected database, restore it to a different workstation, and then catalog and access the database.

Attention: Remove copy-protection from all databases before reinstalling either the database manager or the operating system. If you do not remove copy-protection, you will receive an error when you attempt to access the database. After you have reinstalled, you can enable copy-protection.

DB2 Data Links Manager

The following parameters relate to DB2 Data Links Manager:
- "Data Links Access Token Expiry Interval (dl_expint)"
- "Data Links Number of Copies (dl_num_copies)" on page 415
- "Data Links Time After Drop (dl_time_drop)" on page 415
- "Data Links Token Algorithm (dl_token)" on page 416
- "Data Links Token in Upper Case (dl_upper)" on page 416
- "Enable Data Links Support (datalinks)" on page 416

Data Links Access Token Expiry Interval (dl_expint)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	60 [-1, 1 — 31 536 000]
Unit of Measure	Seconds

This parameter specifies the interval of time (in seconds) for which the generated file access control token is valid. The number of seconds the token is valid begins from the time it is generated. The Data Links Filesystem Filter checks the validity of the token against this expiry time.

For information about file access control tokens, refer to the *DB2 Data Links Manager Quick Beginnings* book.

The default value for this parameter is sixty (60) seconds. Minus one (-1) implies that the token will effectively not expire.

This parameter applies to the DATALINK columns that specify “READ PERMISSION DB”.

Data Links Number of Copies (dl_num_copies)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	0 [0 – 15]

This parameter specifies the number of additional copies of a file to be made in the archive server (such as an ADSM server) when a file is linked to the database.

The default value for this parameter is zero (0).

This parameter applies to the DATALINK columns that specify “Recovery=Yes”.

Data Links Time After Drop (dl_time_drop)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	1 [0 — 365]
Unit of Measure	Days

This parameter specifies the interval of time (in days) files would be retained on an archive server (such as an TSM server) after a DROP DATABASE is issued.

The default value for this parameter is one (1) day. A value of zero (0) means that the files are deleted immediately from the archive server when the DROP command or statement is issued. (The actual file is not deleted unless the ON UNLINK DELETE parameter was specified for the DATALINK column.)

This parameter applies to the DATALINK columns that specify "Recovery=Yes".

Data Links Token Algorithm (dl_token)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	MAC0 [MAC0; MAC1]

This parameter specifies the algorithm used in the generation of DATALINK file access control tokens. The value of MAC1 (message authentication code) generates a more secure message authentication code than MAC0, but also has more performance overhead.

For information about file access control tokens, refer to the *DB2 Data Links Manager Quick Beginnings* book.

This parameter applies to the DATALINK columns that specify "READ PERMISSION DB".

Data Links Token in Upper Case (dl_upper)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	NO [YES; NO]

The parameter indicates whether the file access control tokens use upper case letters. A value of "YES" specifies that all letters in an access control token are upper case. A value of "NO" specifies that the token can contain both upper case and lower case letters.

For information about file access control tokens, refer to the *DB2 Data Links Manager Quick Beginnings* book.

This parameter applies to the DATALINK columns that specify "READ PERMISSION DB".

Enable Data Links Support (datalinks)

Configuration Type	Database manager
Parameter Type	Configurable
Default [Range]	NO [YES; NO]

This parameter specifies whether Data Links support is enabled. A value of "YES" specifies that Data Links support is enabled for Data Links Manager linking files stored in native filesystems (for example, JFS on AIX). A value of "NO" specifies that Data Links support is not enabled.

Status

The following parameters provide information about the state of the database:

- "Backup Pending Indicator (backup_pending)"
- "Database is Consistent (database_consistent)"
- "Roll Forward Pending Indicator (rollfwd_pending)" on page 418
- "Log Retain Status Indicator (log_retain_status)" on page 418
- "User Exit Status Indicator (user_exit_status)" on page 418
- "Restore Pending (restore_pending)" on page 418
- "MultiPage File Allocation Enabled (multipage_alloc)" on page 419

Backup Pending Indicator (backup_pending)

Configuration Type	Database
Parameter Type	Informational

If set on, this parameter indicates that you must do a full backup of the database before accessing it. This parameter is only on if the database configuration is changed so that the database moves from being nonrecoverable to recoverable (that is, initially both the *logretain* and *userexit* parameters were set to NO, then either one or both of these parameters is set to YES, and the update to the database configuration is accepted).

Database is Consistent (database_consistent)

Configuration Type	Database
Parameter Type	Informational

This parameter indicates whether the database is in a consistent state.

YES indicates that all transactions have been committed or rolled back so that the data is consistent. If the system "crashes" while the database is consistent, you do not need to take any special action to make the database usable.

NO indicates that a transaction is pending or some other task is pending on the database and the data is not consistent at this point. If the system "crashes" while the database is not consistent, you will need to restart the database using the RESTART DATABASE command to make the database usable. For more information about the RESTART DATABASE command, see the *Command Reference*.

Roll Forward Pending Indicator (rollfwd_pending)

Configuration Type	Database
Parameter Type	Informational

This parameter can indicate one of the following states:
- **DATABASE**, meaning that a roll-forward recovery procedure is required for this database
- **TABLESPACE**, meaning that one or more table space needs to be rolled forward
- **NO**, meaning that the database is usable and no roll-forward recovery is required.

The recovery (using ROLLFORWARD DATABASE) must complete before you can access the database or table space. For more information about ROLLFORWARD DATABASE, see the *Command Reference*.

Log Retain Status Indicator (log_retain_status)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	“Log Retain Enable (logretain)” on page 399

If set, this parameter indicates that log files are being retained for use in roll-forward recovery.

This parameter is set when the *logretain* parameter setting becomes active.

User Exit Status Indicator (user_exit_status)

Configuration Type	Database
Parameter Type	Informational
Related Parameters	“User Exit Enable (userexit)” on page 399

If set ON, this indicates that the database manager is enabled for roll-forward recovery and that the user exit program will be used to archive and retrieve log files when called by the database manager.

Restore Pending (restore_pending)

Configuration Type	Database
Parameter Type	Informational

This parameter states whether a RESTORE PENDING status exists in the database.

MultiPage File Allocation Enabled (multipage_alloc)

Configuration Type	Database
Parameter Type	Informational

Multipage file allocation is used to improve insert performance. It applies to SMS table spaces only. If enabled, all SMS table spaces are affected: there is no selection possible for individual SMS table spaces.

The default for the parameter is NO: multipage file allocation is not enabled.

Following database creation, the parameter may be set to YES which indicates that multipage file allocation is enabled. This is done using the *db2empfa* tool. Once set to YES, the parameter cannot be changed back to NO.

Compiler Settings

The following parameters provide information to influence the compiler:

- "Continue upon Arithmetic Exceptions (dft_sqlmathwarn)"
- "Default Degree (dft_degree)" on page 421
- "Default Query Optimization Class (dft_queryopt)" on page 421
- "Default Refresh Age (dft_refresh_age)" on page 422
- "Number of Frequent Values Retained (num_freqvalues)" on page 422
- "Number of Quantiles for Columns (num_quantiles)" on page 423

Continue upon Arithmetic Exceptions (dft_sqlmathwarn)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [No, Yes]

This parameter sets the default value that determines the handling of arithmetic errors and retrieval conversion errors as errors or warnings during SQL statement compilation. For static SQL statements, the value of this parameter is associated with the package at bind time. For dynamic SQL DML statements, the value of this parameter is used when the statement is prepared.

Attention: If you change the *dft_sqlmathwarn* value for a database, the behavior of check constraints, triggers, and views that include arithmetic expressions may change. This may in turn have an impact on the data integrity of the database. You should only change the setting of *dft_sqlmathwarn* for a database after carefully evaluating how the new arithmetic exception handling behavior may impact check constraints, triggers, and views. Once changed, subsequent changes require the same careful evaluation.

As an example, consider the following check constraint, which includes a division arithmetic operation:

```
A/B > 0
```

When *dft_sqlmathwarn* is "No" and an INSERT with B=0 is attempted, the division by zero is processed as an arithmetic error. The insert operation fails because DB2 cannot check the constraint. If *dft_sqlmathwarn* is changed to "Yes", the division by zero is processed as an arithmetic warning with a NULL result. The NULL result causes the ">" predicate to evaluate to UNKNOWN and the insert operation succeeds. If *dft_sqlmathwarn* is changed back to "No", an attempt to insert the same row will fail, because the division by zero error prevents DB2 from evaluating the constraint. The row inserted with B=0 when *dft_sqlmathwarn* was "Yes" remains in the table and can be selected. Updates to the row that cause the constraint to be evaluated will fail, while updates to the row that do not require constraint re-evaluation will succeed.

Before changing *dft_sqlmathwarn* from "No" to "Yes", you should consider rewriting the constraint to explicitly handle nulls from arithmetic expressions. For example:

```
  ( A/B > 0 ) AND ( CASE
                      WHEN A IS NULL THEN 1
                      WHEN B IS NULL THEN 1
                      WHEN A/B IS NULL THEN 0
                      ELSE 1
                      END
                    = 1 )
```

can be used if both A and B are nullable. And, if A or B is not-nullable, the corresponding IS NULL WHEN-clause can be removed.

Before changing *dft_sqlmathwarn* from "Yes" to "No", you should first check for data that may become inconsistent, for example by using predicates such as the following:

```
   WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

When inconsistent rows are isolated, you should take appropriate action to correct the inconsistency before changing *dft_sqlmathwarn*. You can also manually re-check constraints with arithmetic expressions after the change. To do this, first place the affected tables in a check pending state (with the OFF clause of the SET CONSTRAINTS statement), then request that the tables be checked (with the IMMEDIATE CHECKED clause of the SET CONSTRAINTS statement). Inconsistent data will be indicated by an arithmetic error, which prevents the constraint from being evaluated.

Recommendation: Use the default setting of no, unless you specifically require queries to be processed that include arithmetic exceptions. Then specify the value of yes. This situation can occur if you are processing SQL statements that, on other database managers, provide results regardless of the arithmetic exceptions that occur.

Default Degree (dft_degree)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	1 [-1, 1 – 32 767]
Related Parameters	"Maximum Query Degree of Parallelism (max_querydegree)" on page 443

This parameter specifies the default value for the CURRENT DEGREE special register and the DEGREE bind option.

The default value is 1.

A value of 1 means no intra-partition parallelism. A value of -1 means the optimizer determines the degree of intra-partition parallelism based on the number of processors and the type of query.

The degree of intra-partition parallelism for a SQL statement is specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option. The maximum runtime degree of intra-partition parallelism for an active application is specified using the SET RUNTIME DEGREE command. The Maximum Query Degree of Parallelism (max_querydegree) configuration parameter specifies the maximum query degree of intra-partition parallelism for all SQL queries.

The actual runtime degree used is the lowest of:
- *max_querydegree* configuration parameter
- application runtime degree
- SQL statement compilation degree

Default Query Optimization Class (dft_queryopt)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	5 [0 — 9]
Unit of Measurement	Query Optimization Class (see below)

The query optimization class is used to direct the optimizer to use different degrees of optimization when compiling SQL queries. This parameter provides additional flexibility by setting the default query optimization class used when neither the SET CURRENT QUERY OPTIMIZATION statement nor the QUERYOPT bind command are used.

The query optimization classes currently defined are:

0 - minimal query optimization.
1 - roughly comparable to DB2 Version 1.
2 - slight optimization.
3 - moderate query optimization.
5 - significant query optimization with heuristics to limit the effort expended on selecting an access plan. This is the default.
7 - significant query optimization.
9 - maximal query optimization

Recommendation: For more information and guidance for selecting a suitable query optimization class, see “Adjusting the Optimization Class” on page 64.

For more information on how a program can retrieve and modify database configuration parameters, see the *Administrative API Reference*.

Default Refresh Age (dft_refresh_age)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	0 [0, 99999999999999 (ANY)]

This parameter has the default value used for the REFRESH AGE is the CURRENT REFRESH AGE special register is not specified. This parameter specifies a time stamp duration value with a data type of DECIMAL(20,6). This time duration represents the maximum duration since a REFRESH TABLE statement has been processed on a specific REFRESH DEFERRED summary table during which that summary table can be used to optimize the processing of a query. If the CURRENT REFRESH AGE has a value of 99999999999999 (ANY), and the QUERY OPTIMIZATION class is five or more, REFRESH DEFERRED summary tables are considered to optimize the processing of a dynamic SQL query.

Number of Frequent Values Retained (num_freqvalues)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	10 [0 — 32 767]
Unit of Measure	Counter

Related Parameters	• "Number of Quantiles for Columns (num_quantiles)" • "Statistics Heap Size (stat_heap_sz)" on page 345

This parameter allows you to specify the number of "most frequent values" that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command. Increasing the value of this parameter increases the amount of statistics heap (*stat_heap_sz*) used when collecting statistics.

The "most frequent value" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the SQL optimizer but requires additional catalog space. When 0 is specified, no frequent-value statistics are retained, even if you request that distribution statistics be collected.

Updating this parameter can help the optimizer obtain better selectivity estimates for some predicates (=, <, >, IS NULL, IS NOT NULL) over data that is non-uniformly distributed. More accurate selectivity calculations may result in the choice of more efficient access plans.

After changing the value of this parameter, you need to:
- Run the RUNSTATS command after all users have disconnected from the database and you have reconnected to the database
- Rebind any packages containing static SQL.

For more information, see "Collecting and Using Distribution Statistics" on page 116.

Recommendation: In order to update this parameter you should determine the degree of non-uniformity in the most important columns (in the most important tables) that typically have selection predicates. This can be done using an SQL SELECT statement that provides an ordered ranking of the number of occurrences of each value in a column. You should not consider uniformly distributed, unique, long, or LOB columns. A reasonable practical value for this parameter lies in the range of 10 to 100.

Note that the process of collecting frequent value statistics requires significant CPU and memory (*stat_heap_sz*) resources.

Number of Quantiles for Columns (num_quantiles)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	20 [0 – 32 767]

Unit of Measure	Counter
Related Parameters	• "Number of Frequent Values Retained (num_freqvalues)" on page 422 • "Statistics Heap Size (stat_heap_sz)" on page 345

This parameter controls the number of quantiles that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command. Increasing the value of this parameter increases the amount of statistics heap (*stat_heap_sz*) used when collecting statistics.

The "quantile" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the SQL optimizer but requires additional catalog space. When 0 or 1 is specified, no quantile statistics are retained, even if you request that distribution statistics be collected.

Updating this parameter can help obtain better selectivity estimates for range predicates over data that is non-uniformly distributed. Among other optimizer decisions, this information has a strong influence on whether an index scan or a table scan will be chosen. (It is more efficient to use a table scan to access a range of values that occur frequently and it is more efficient to use an index scan for a range of values that occur infrequently.)

After changing the value of this parameter, you need to:
- Run the RUNSTATS command after all users have disconnected from the database and you have reconnected to the database
- Rebind any packages containing static SQL.

For more information, see "Collecting and Using Distribution Statistics" on page 116.

Recommendation: This default value for this parameter guarantees a maximum estimation error of approximately 2.5% for any single-sided range predicate (>, >=, <, or <=), and a maximum error of 5% for any BETWEEN predicate. A rough rule of thumb for determining the number of quantiles is:
- Determine the maximum error that is tolerable in estimating the number of rows of any range query, as a percentage, P
- The number of quantiles should be approximately 100/P if most of your predicates are BETWEEN predicates, and 50/P if most of your predicates are other types of range predicates (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and of 2% for ″>″ predicates. A reasonable practical value for this parameter lies in the range of 10 to 50.

Communications

The following groups of parameters provide information about using DB2 in a client/server environment:

- "Communication Protocol Setup"
- "Distributed Services" on page 429
- "DB2 Discovery" on page 434

Communication Protocol Setup

You can use the following parameters to configure your database clients and database servers:

- "NetBIOS Workstation Name (nname)"
- "TCP/IP Service Name (svcename)" on page 426
- "APPC Transaction Program Name (tpname)" on page 427
- "IPX/SPX File Server Name (fileserver)" on page 427
- "IPX/SPX DB2 Server Object Name (objectname)" on page 428
- "IPX/SPX Socket Number (ipx_socket)" on page 429

NetBIOS Workstation Name (nname)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

This parameter allows you to assign a unique name to the database instance on a workstation in the NetBIOS LAN environment. This *nname* is the basis for the actual NetBIOS names that will be registered with NetBIOS for a workstation.

Since the NetBIOS protocol establishes connections using these NetBIOS names, the *nname* parameter must be set for both the client and server.

Client applications must know the *nname* of the server that contains the database to be accessed. The server's *nname* must be cataloged in the client's node directory as the "server-nname" parameter using the CATALOG NETBIOS NODE command.

If *nname* at the server node changes to a new name, all clients accessing databases on that server must catalog this new name for the server.

TCP/IP Service Name (svcename)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default	Null

This parameter contains the name of the TCP/IP port which a database server will use to await communications from remote client nodes. This name must be the first of two consecutive ports reserved for use by the database manager; the second port is used to handle interrupt requests from down-level clients.

In order to accept connection requests from a database client using TCP/IP, the database server must be listening on a port designated to that server. The system administrator for the database server must reserve a port (number *n*) and define its associated TCP/IP service name in the services file at the server. If the database server needs to support requests from down-level clients, a second port (number *n+1*, for interrupt requests) needs to be defined in the services file at the server.

The database server port (number *n*) and its TCP/IP service name need to be defined in the services file on the database client. Down-level clients also require the interrupt port (number *n+1*) to be defined in the client's services file.

The location of the services file depends on your operating environment. For example:
- In UNIX — /etc/services
- In OS/2 — \tcpip\etc\services
- In OS/2 Warp — \mptn\etc\services.

The *svcename* parameter should be set to the service name associated with the main connection port so that when the database server is started, it can determine on which port to listen for incoming connection requests. If you are supporting or using a down-level client, the service name for the interrupt port is not saved in the configuration file. The interrupt port number can be derived based on the main connection port number (*interrupt port number* = *main connection port* + 1).

Refer to the *Installation and Configuration Supplement* for more information about setting up TCP/IP for database servers.

APPC Transaction Program Name (tpname)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

This parameter defines the name of the remote transaction program that the database client must use when it issues an allocate request to the database serverwhen using the APPC communication protocol. This parameter must be set in the configuration file at the database server.

This parameter must be the same as the transaction program name that is configured in the SNA transaction program definition. Refer to the *Installation and Configuration Supplement* for more information about setting up APPC for your DB2 product.

Recommendation: The only accepted characters for use in this name are:
- Alphabetics (A through Z; or a through z)
- Numerics (0 through 9)
- Dollar sign ($), number sign (#), at sign (@), and period (.)

IPX/SPX File Server Name (fileserver)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients

	• Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null
Related Parameters	• "IPX/SPX DB2 Server Object Name (objectname)" • "IPX/SPX Socket Number (ipx_socket)" on page 429

This parameter specifies the name of the NetWare** fileserver where the internetwork address of the database manager is registered. The internetwork address of the database manager is stored in the bindery at the NetWare file server. If the registered fileserver name changes, all clients that access the server instance must:
- UNCATALOG the server node
- CATALOG the server node, specifying the new fileserver name.

For more information, refer to the *Installation and Configuration Supplement*.

IPX/SPX DB2 Server Object Name (objectname)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null
Related Parameters	• "IPX/SPX File Server Name (fileserver)" on page 427 • "IPX/SPX Socket Number (ipx_socket)" on page 429

This parameter provides the name of the database manager instance in an IPX/SPX network. Each server instance registered to a NetWare fileserver must have a unique name. If this name changes at the database server, all clients that access the server must uncatalog the server node and recatalog it again, specifying the new object name.

IPX/SPX Socket Number (ipx_socket)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	879E [879E – 87A2] To ensure that there are no conflicts, five socket numbers (879E to 87A2) are uniquely registered with Novell for use by DB2.
Related Parameters	• "IPX/SPX File Server Name (fileserver)" on page 427 • "IPX/SPX DB2 Server Object Name (objectname)" on page 428

This parameter specifies a "well-known" socket number and represents the connection end point in a DB2 server's internetwork address. The socket number must be unique for each DB2 server instance on a given machine, and unique among all Novell** IPX/SPX applications running on this same machine. This is to guarantee that the DB2 server is able to listen to incoming IPX/SPX connections using this socket number.

Distributed Services

You can use the following parameters to configure your database clients and database servers to make use of DCE Directory services:

- "Directory Services Type (dir_type)" on page 430
- "Directory Path Name in DCE Namespace (dir_path_name)" on page 431
- "Object Name in DCE Namespace (dir_obj_name)" on page 431
- "Routing Information Object Name (route_obj_name)" on page 432
- "Default Client Communication Protocol (dft_client_comm)" on page 433
- "Default Client Adapter Number (dft_client_adpt)" on page 434

For information about how DB2 uses DCE directories, refer to “Using Distributed Computing Environment (DCE) Directory Services” in *Administration Guide: Planning*.

Directory Services Type (dir_type)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • UNIX and OS/2 Client • UNIX and OS/2 Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	NONE [NONE; DCE]
Related Parameters	• “Object Name in DCE Namespace (dir_obj_name)” on page 431 • “Directory Path Name in DCE Namespace (dir_path_name)” on page 431 • “Routing Information Object Name (route_obj_name)” on page 432 • “Default Client Communication Protocol (dft_client_comm)” on page 433 • “Default Client Adapter Number (dft_client_adpt)” on page 434

This parameter indicates whether or not DCE directory services is used.

If this parameter is set to **NONE**, only local directory files will be searched for the target of the CONNECT or ATTACH requests. However, you can still use the *dir_path_name* and *dir_obj_name* parameters to record the name of your database instance and databases in the DCE namespace.

If this parameter is set to **DCE**, then when an application running within this database manager instance cannot find the target of its CONNECT or ATTACH requests, the DCE directory will be searched.

For the numeric equivalents and API constants for these values, refer to the *Administrative API Reference*.

Directory Path Name in DCE Namespace (dir_path_name)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • UNIX and OS/2 Client • UNIX and OS/2 Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	/.:/subsys/database/
Related Parameters	• "Object Name in DCE Namespace (dir_obj_name)" • "Directory Services Type (dir_type)" on page 430 • "Routing Information Object Name (route_obj_name)" on page 432

The unique name of the database manager instance in the global namespace is made up of this value and the value in the *dir_obj_name* parameter.

All client applications running within this instance also use it as the default path name for their CONNECT or ATTACH requests, unless it is overridden by the value of the DB2DIRPATHNAME environment variable.

Recommendation: Use the name provided by your DCE administrator.

Object Name in DCE Namespace (dir_obj_name)

Configuration Type	Database manager, Database
Applies to	• Database server with local and remote clients • UNIX and OS/2 Client • UNIX and OS/2 Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

Related Parameters	• "Directory Services Type (dir_type)" on page 430 • "Directory Path Name in DCE Namespace (dir_path_name)" on page 431

The object name representing your database manager instance (or your database) in the directory. The concatenation of this value and the *dir_path_name* value yields a global name that uniquely identifies the database manager instance or database in the namespace governed by the directory services specified in the *dir_type* parameter.

This parameter is only meaningful if the *dir_path_name* parameter is specified.

The total length of the configuration parameters *dir_path_name* and *dir_obj_name* must be less than 255 characters.

Recommendation: Refer to "Using Distributed Computing Environment (DCE) Directory Services" in the *Administration Guide: Implementation* for more information.

Routing Information Object Name (route_obj_name)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null
Related Parameters	• "Directory Path Name in DCE Namespace (dir_path_name)" on page 431 • "Directory Services Type (dir_type)" on page 430

This parameter specifies the name of the default routing information object entry that will be used by all client applications attempting to access a DRDA server. It applies to OS/2 and UNIX-based environments only.

If the value of this parameter starts with /.:/ or /.../, then the value will be used as is. Otherwise, it will be appended to the *dir_path_name* parameter (or DB2DIRPATHNAME environment variable) value to form the full name of the routing information object.

You can use the environment variable DB2ROUTE to override this default.

This parameter is only meaningful if the *dir_type* parameter is set to DCE.

Recommendation: Refer to "Using Distributed Computing Environment (DCE) Directory Services" in the *Administration Guide: Implementation* for more information.

Default Client Communication Protocol (dft_client_comm)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [Null; TCPIP; APPC; IPXSPX (OS/2 only); NETBIOS (OS/2 only)]
Related Parameters	"Directory Services Type (dir_type)" on page 430

This parameter indicates the communication protocols that the client applications on this instance can use for remote connections. Its content is a character string, made up of one or more tokens. If you are specifying more than one token, separate them with a comma. The order of the tokens is significant in terms of preference.

This parameter can only be used with DCE, and applies to OS/2 and UNIX-based environments only.

You can temporarily override the value of this parameter by setting the DB2CLIENTCOMM environment variable.

If the value of this parameter is NULL and the environment variable has not been set, the first protocol specified in the server's global directory object is used.

This parameter is ignored if *dir_type* is set to NONE.

Recommendation: The protocol that is used most often should be specified first.

Default Client Adapter Number (dft_client_adpt)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Client • Database server with local clients
Parameter Type	Configurable
Default [Range]	0 [0–15]
Related Parameters	• "Default Client Communication Protocol (dft_client_comm)" on page 433. • "Directory Services Type (dir_type)" on page 430. (When dir_type is set to DCE.)

This parameter defines the default client adapter number for the NETBIOS protocol whose server nname is extracted from DCE Cell Directory Services (CDS). This parameter is applicable to the OS/2 environment only.

This parameter can only be used with DCE.

You can temporarily override the value of this parameter by setting the DB2CLIENTADPT environment variable. If this environment variable contains a non-numeric or out-of-range number, adapter number 0 (zero) is used.

DB2 Discovery

You can use the following parameters to establish DB2 Discovery:

- "Discover Database (discover_db)"
- "Discovery Mode (discover)" on page 435
- "Search Discovery Communications Protocols (discover_comm)" on page 436
- "Discover Server Instance (discover_inst)" on page 437

Discover Database (discover_db)

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Enable [Disable, Enable]

This parameter is used to prevent information about a database from being returned to a client when a discovery request is received at the server.

The default for this parameter is that discovery is enabled for this database.

By changing this parameter value to "Disable", it is possible to hide databases with sensitive data from the discovery process. This can be done in addition to other database security controls on the database.

For the numeric equivalents and API constants for these values, refer to the *Administrative API Reference*.

Discovery Mode (discover)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	search [disable, known, search]
Related Parameters	"Search Discovery Communications Protocols (discover_comm)" on page 436

From an administration server perspective, this configuration parameter determines the type of discovery mode that is started when DB2ADMIN starts.

- If *discover* = SEARCH when DB2ADMIN starts, then search discovery connection managers for each of the protocols specified in *discover_comm* are started. In addition, connection managers for each of the protocols specified in the DB2COMM registry variable are started. This allows the administration server to handle search discovery requests from clients. SEARCH provides a superset of the functionality provided by KNOWN discovery. When *discover* = SEARCH, the administration server will handle both search and known discovery requests from clients.
- If *discover* = KNOWN when DB2ADMIN starts, then only the connection managers specified in the DB2COMM registry variable are started. These connection managers handle KNOWN discovery requests.

- If *discover* = DISABLE when DB2ADMIN starts, then the administration server will not handle any type of discovery request.

From a server instance perspective, if *discover* = DISABLE then the information for this server instance is essentially hidden from clients. The administration server will not package information about this instance when a known discovery request is issued against this system by any client.

From a client perspective, one of the following will occur:

- If *discover* = SEARCH, the client can issue search discovery requests to find DB2 server systems on the network. Search discovery provides a superset of the functionality provided by KNOWN discovery. If *discover* = SEARCH, both search and known discovery requests can be issued by the client.
- If *discover* = KNOWN, only known discovery requests can be issued from the client. By specifying some connection information for the administration server on a particular system, all the instance and database information on the DB2 system is returned to the client.
- If *discover* = DISABLE, discovery is disabled at the client.

The default discovery mode is SEARCH.

For the numeric equivalents and API constants for these values, refer to the *Administrative API Reference*.

For more information on DB2 Discovery, refer to the *Quick Beginnings* manual appropriate to your platform.

Search Discovery Communications Protocols (discover_comm)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	None [Any combination of NETBIOS and TCPIP]
Related Parameters	“Discovery Mode (discover)” on page 435

From an administration server perspective this parameter defines the search discovery managers that are started when DB2ADMIN starts. These managers service search discovery requests from clients.

Note: The protocols defined in *discover_comm* must also be specified in the DB2COMM registry variable.

From a client perspective, this parameter defines the protocols that clients use to issue search discovery requests.

More than one protocol may be specified, separated by commas; or, the parameter may be left blank.

The default for this parameter is ″None″ meaning that there are no search discovery communications protocols.

Discover Server Instance (discover_inst)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	enable [enable, disable]

This parameter specifies whether this instance can be detected by DB2 Discovery. The default, "enable", specifies that the instance can be detected, while "disable" prevents the instance from being discovered.

For the numeric equivalents and API constants for these values, refer to the *Administrative API Reference*.

For more information on DB2 Discovery, refer to the *Quick Beginnings* manual appropriate to your platform.

Parallel

The following groups of parameters provide information about parallel operations and partitioned database environments:
- "Communications" on page 438
- "Parallel Processing" on page 443.

Communications

The following parameters provide information about communications in the partitioned databaseenvironment:

- "Connection Elapse Time (conn_elapse)"
- "Number of FCM Message Anchors (fcm_num_anchors)"
- "Number of FCM Buffers (fcm_num_buffers)" on page 439
- "Number of FCM Connection Entries (fcm_num_connect)" on page 440
- "Number of FCM Request Blocks (fcm_num_rqb)" on page 441
- "Node Connection Retries (max_connretries)" on page 442
- "Maximum Time Difference Among Nodes (max_time_diff)" on page 442
- "Start and Stop Timeout (start_stop_time)" on page 443.

Connection Elapse Time (conn_elapse)

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	10 [0–100]
Unit of Measure	Seconds
Related Parameters	"Node Connection Retries (max_connretries)" on page 442

This parameter specifies the number of seconds within which a TCP/IP connection is to be established between two database partition servers. If the attempt completes within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the *max_connretries* parameter and always times out, an error is issued.

Number of FCM Message Anchors (fcm_num_anchors)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients

Parameter Type	Configurable
Default [Range]	-1 [-1, 128–*fcm_num_rqb*] On non-partitioned database systems, *intra_parallel* parameter must be active before this parameter can be used.
Related Parameters	• "Number of FCM Request Blocks (fcm_num_rqb)" on page 441 • "Enable Intra-Partition Parallelism (intra_parallel)" on page 445

This parameter specifies the number of FCM *message anchors*. Agents use the message anchors to send messages among themselves. The default (-1) indicates 75 percent of the value specified for *fcm_num_rqb*.

Number of FCM Buffers (fcm_num_buffers)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	512, 1 024, or 4 096 [128 — *fcm_num_rqb*] • Database server with local and remote clients: the default is 1 024 • Database server with local clients: the default is 512 • Partitioned database server with local and remote clients: the default is 4 096 On single-partition database systems, the *intra_parallel* parameter must be active before this parameter can be used.

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within the database servers in a partitioned database environment.

Refer to “Enable FCM Communications” in the *Administration Guide: Implementation* for more information on FCM.

If you have multiple logical nodes on a processor, you may find it necessary to increase the value of this parameter. You may also find it necessary to increase the value of this parameter if you run out of message buffers because of the number of users on the system, the number of database partition servers on the system, or the complexity of the applications.

If you are using multiple logical nodes, on non-AIX systems, one pool of *fcm_num_buffers* buffers is shared by all the multiple logical nodes on the same machine, while on AIX:

- If there is enough room in the general memory that is used by the database manager, the FCM buffer heap will be allocated from there. In this situation, each database partition server will have *fcm_num_buffers* buffers of its own; the database partition servers will not share a pool of FCM buffers (this was new in DB2 Version 5).
- If there is not enough room in the general memory that is used by the database manager, the FCM buffer heap will be allocated from a separate memory area (AIX shared memory set), that is shared by all the multiple logical nodes on the same machine. One pool of *fcm_num_buffers* will be shared by all the multiple logical nodes on the same machine. This is the same as non-AIX systems and is also the same as DB2 Parallel Edition Version 1.2 on AIX.

Recommendation for existing Parallel Edition customers on AIX: If you are using multiple logical nodes, the value of *fcm_num_buffers* you used in Parallel Edition Version 1.2 may now result in significantly more storage being used per machine. For example, a four-node multiple logical node configuration may end up with four times as many FCM buffers as before.

Re-examine the value you are using; consider how many FCM buffers in total will be allocated on the machine (or machines) where the multiple logical nodes reside. You may want to change *fcm_num_buffers* to account for the behavior described above.

Number of FCM Connection Entries (fcm_num_connect)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients

- Satellite database server with local clients

Parameter Type	Configurable
Default [Range]	-1 [-1, 128 — *fcm_num_rqb*] On non-partitioned database systems, the *intra_parallel* parameter must be active before this parameter can be used.
Related Parameters	"Number of FCM Request Blocks (fcm_num_rqb)"

This parameter specifies the number of FCM *connection entries*. Agents use connection entries to pass data among themselves. The default (-1) indicates 75 percent of the value specified for *fcm_num_rqb*.

Number of FCM Request Blocks (fcm_num_rqb)

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type Configurable

Default [Range]

UNIX 32-bit platforms
256, 512, 2 048 [128 — 120 000]

UNIX 64-bit platforms
256, 512, 2 048 [128 — 524 288]

OS/2 and Windows NT
10 000 [250 — 2 097 152]

- Database server with local and remote clients: the default is 512
- Database server with local clients: the default is 256
- Partitioned database server with local and remote clients: the default is 2 048

	On non-partitioned database systems, the *intra_parallel* parameter must be active before this parameter can be used.

This parameter specifies the number of FCM *request blocks*. Request blocks are the media through which information is passed between the FCM daemon and an agent, or between agents.

The requirement for request blocks will vary according to the number of users on the system, the number of database partition servers in the system, and the complexity of queries that are run. Initially, start with the default number, and use the results from the Database System Monitor when fine tuning this parameter.

Node Connection Retries (max_connretries)

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	5 [0–100]
Related Parameters	"Connection Elapse Time (conn_elapse)" on page 438

If the attempt to establish communication between two database partition servers fails (for example, the value specified by the *conn_elapse* parameter is reached), *max_connretries* specifies the number of connection retries that can be made to a database partition server. If the value specified for this parameter is exceeded, an error is returned.

Maximum Time Difference Among Nodes (max_time_diff)

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	60 [1–1 440]
Unit of Measure	Minutes

Each database partition server has its own system clock. This parameter specifies the maximum time difference, in minutes, that is permitted among the database partition servers listed in the node configuration file.

If two or more database partition servers are associated with a transaction and their clocks are not synchronized to within the time specified by this parameter, the transaction is rejected and a warning or an error message is logged in the `db2diag.log` file. (The transaction is rejected only if data modification is associated with it.)

DB2 Universal Database Enterprise - Extended Edition uses *Coordinated Universal Time*, (UTC) so different time zones are not a consideration when you set this parameter. The Coordinated Universal Time is the same as Greenwich Mean Time.

Start and Stop Timeout (start_stop_time)

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	10 [1 — 1 440]
Unit of Measure	Minutes

This parameter is applicable in a partitioned database environment only. It specifies the time, in minutes, within which all database partition servers must respond to a DB2START or a DB2STOP command. It is also used as the timeout value during an ADDNODE operation.

Database partition servers that do not respond to a DB2START command within the specified time send a message to the `db2start` error log in the `log` subdirectory of the `sqllib` subdirectory of the home directory for the instance. You should issue a DB2STOP on these nodes before restarting them.

Database partition servers that do not respond to a DB2STOP command within the specified time send a message to the `db2stop` error log in the `log` subdirectory of the `sqllib` subdirectory of the home directory for the instance. You can either issue DB2STOP for each database partition server that does not respond, or for all of them. (Those that are already stopped will return stating that they are stopped.)

Parallel Processing

The following parameters provide information about parallel processing:

- "Maximum Query Degree of Parallelism (max_querydegree)"
- "Enable Intra-Partition Parallelism (intra_parallel)" on page 445.

Maximum Query Degree of Parallelism (max_querydegree)

Configuration Type	Database manager

Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	-1 (ANY) [ANY, 1 — 32 767] (ANY means system determined)
Related Parameters	• "Default Degree (dft_degree)" on page 421 • "Enable Intra-Partition Parallelism (intra_parallel)" on page 445

This parameter specifies the maximum degree of intra-partition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a partition when the statement is executed. The *intra_parallel* configuration parameter must be set to "YES" to enable the database partition to use intra-partition parallelism.

The default value for this configuration parameter is -1. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

Note: The degree of parallelism for an SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option.

The maximum query degree of parallelism for an active application can be modified using the SET RUNTIME DEGREE command. The actual runtime degree used is the lower of:
- *max_querydegree* configuration parameter
- Application runtime degree
- SQL statement compilation degree

An exception regarding the determination of the actual query degree of parallelism occurs when creating an index. In this case, if *intra_parallel* is "YES" and the table is large enough to benefit from the use of multiple processors, then creating an index uses the number of online processors (to a maximum of 6) plus one. There is no effect from the other parameter, bind option, or special register mentioned above.

Enable Intra-Partition Parallelism (intra_parallel)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	NO (0) [SYSTEM (-1), NO (0), YES (1)] A value of -1 causes the parameter value to be set to "YES" or "NO" based on the hardware on which the database manager is running.
Related Parameters	"Maximum Query Degree of Parallelism (max_querydegree)" on page 443

This parameter specifies whether the database manager can use intra-partition parallelism.

Some of the operations that can take advantage of parallel performance improvements when this parameter is ″YES″ include database queries and index creation.

Note: If you change this parameter value, packages may be rebound to the database. If this occurs, a performance degradation may occur during the rebinding.

Instance Management

A number of parameters can help you manage your database managerinstances. These are grouped into the following categories:
- "Diagnostic"
- "Database System Monitor Parameters" on page 448
- "System Management" on page 450
- "Instance Administration" on page 457

Diagnostic

The following parameters allow you to control diagnostic information available from the database manager:
- "Diagnostic Error Capture Level (diaglevel)" on page 446

- "Diagnostic Data Directory Path (diagpath)"
- "Notify Level (notifylevel)" on page 447.

Diagnostic Error Capture Level (diaglevel)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	3 [0 — 4]
Related Parameters	"Diagnostic Data Directory Path (diagpath)"

The type of diagnostic errors recorded in the db2diag.log file is determined by this parameter. Valid values are:

0 – No diagnostic data captured
1 – Severe errors only
2 – All errors
3 – All errors and warnings
4 – All errors, warnings and informational messages

It is the *diagpath* configuration parameter that is used to specify the directory that will contain the error file, event log file (on Windows NT only), alert log file, and any dump files that may be generated based on the value of the *diaglevel* parameter.

Recommendation: You may wish to increase the value of this parameter to gather additional problem determination data to help resolve a problem.

Diagnostic Data Directory Path (diagpath)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients

	• Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	Null [any valid path name]
Related Parameters	"Diagnostic Error Capture Level (diaglevel)" on page 446

This parameter allows you to specify the fully qualified path for DB2 diagnostic information. This directory could possibly contain dump files, trap files, an error log and an alert log file, depending on your platform.

If this parameter is null, the diagnostic information will be written to files in one of the following directories or folders:
- For OS/2 and supported Windows environments:
  - If the DB2INSTPROF environment variable or keyword is **not** set, information will be written to `x:\SQLLIB\DB2INSTANCE`, where `x:\SQLLIB` is the drive reference and directory specified in the DB2PATH registry variable or environment variable. and `DB2INSTANCE` is the name of the instance owner.

    Note: The directory does not have to be named SQLLIB.
  - If the DB2INSTPROF environment variable or keyword is set, information will be written to `x:\DB2INSTPROF\DB2INSTANCE`, where DB2INSTPROF is the name of the instance profile directory and DB2INSTANCE is the name of the instance owner.
- For UNIX-based environments: `INSTHOME/sqllib/db2dump`, where `INSTHOME` is the home directory of the instance owner.
- For Macintosh environments: DB2 folder.

Recommendation: Use the default or have a centralized location for the diagpath of multiple instances.

In a multinode environment, the path you specify must reside on a shared file system.

Notify Level (notifylevel)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients on Windows NT • Client on Windows NT • Database server with local clients on Windows NT

- Partitioned database server with local and remote clients on Windows NT
- Satellite database server with local clients on Windows 95, Windows 98, and Windows NT

Parameter Type	Configurable
Default [Range]	2 [0 — 4]

This parameter specifies the type of administration notification error messages that are written to a file. For a server of the satellite node type, errors are written to the notification file called `*instance*.nfy`. For all other node types, this parameter is only available on the Windows NT platform, and errors are written to the Windows NT event log. The errors can be written by DB2, the Capture and Apply programs, and user applications.

Valid values for this parameter are:

0 — No diagnostic data captured

1 — Severe errors only

2 — All errors

3 — All errors and warnings

4 — All errors, warnings, and informational messages

For a user application to be able to write to the notification file or Windows NT event log, it must call the db2AdminMsgWrite API. For more information about this API, refer to the *Administrative API Reference*.

Recommendation: You may wish to increase the value of this parameter to gather additional problem determination data to help resolve a problem.

Database System Monitor Parameters

The following parameter allows you to control various aspects of the database system monitor:

- "Default Database System Monitor Switches (dft_monswitches)"

Default Database System Monitor Switches (dft_monswitches)

Configuration Type	Database manager
Applies to	

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type	Configurable
Default	All switches turned off

This parameter is unique in that it allows you to set a number of switches which are each internally represented by a bit of the parameter. Depending on the interface you are using to update the database manager configuration, you may be able to update this parameter directly. You may also update each of these switches independently by setting the following parameters:

dft_mon_uow	Default value of the snapshot monitor's unit of work (UOW) switch
dft_mon_stmt	Default value of the snapshot monitor's statement switch
dft_mon_table	Default value of the snapshot monitor's table switch
dft_mon_bufpool	Default value of the snapshot monitor's buffer pool switch
dft_mon_lock	Default value of the snapshot monitor's lock switch
dft_mon_sort	Default value of the snapshot monitor's sort switch

Changes to any of these database system monitor switches take effect immediately; that is, you do not have to stop and restart the database manager.

Note: An existing monitoring application will not automatically use the new default value for a switch. To use the new value (or values), the application must terminate and re-attach to the instance.

For more information about the snapshot monitor and how it uses monitor switches, see the *System Monitor Guide and Reference*.

Recommendation: Any switch that is turned ON instructs the database manager to collect monitor data related to that switch. Collecting additional monitor data increases database manager overhead which can impact system performance.

All monitoring applications inherit these default switch settings when the application issues its first monitoring request (for example, setting a switch, activating the event monitor, taking a snapshot). You should turn on a switch in the configuration file only if you want to collect data starting from the moment the database manager is started. (Otherwise, each monitoring application can set its own switches and the data it collects becomes relative to the time its switches are set.)

System Management

The following parameters relate to system management:

- "Communications Bandwidth (comm_bandwidth)"
- "CPU Speed (cpuspeed)" on page 451
- "Maximum Number of Concurrently Active Databases (numdb)" on page 451
- "Transaction Processor Monitor Name (tp_mon_name)" on page 453
- "Machine Node Type (nodetype)" on page 455
- "Default Charge-Back Account (dft_account_str)" on page 456
- "Java Development Kit 1.1 Installation Path (jdk11_path)" on page 456
- "Federated Database System Support (federated)" on page 457.

Communications Bandwidth (comm_bandwidth)

Configuration Type	Database manager
Applies to	Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	-1 [.1 – 100 000] A value of -1 causes the parameter value to be reset to the default. The default value is calculated based on whether a high speed switch is being used.
Unit of Measure	Megabytes per second

The value calculated for the communications bandwidth, in megabytes per second, is used by the SQL optimizer to estimate the cost of performing certain operations between the database partition servers of a partitioned database system. The optimizer does not model the cost of communications between a client and a server, so this parameter should reflect only the nominal bandwidth between the database partition servers, if any.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The communications bandwidth is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

CPU Speed (cpuspeed)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	-1 [1e-10 — 1] A value of -1 will cause the parameter value to be reset based on the running of the measurement program.

The CPU speed, in milliseconds per instruction, is used by the SQL optimizer to estimate the cost of performing certain operations. The value of this parameter is set automatically when you install the database manager based on the output from a program designed to measure CPU speed. This program is executed, if benchmark results are not available for any of the following reasons:
- The platform does not have support for the `db2spec.dat` file
- The `db2spec.dat` file is **not** found
- The data for the IBM RISC System/6000 model 530H is not found in the file
- The data for your machine is not found in the file.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware. By setting it to -1, *cpuspeed* will be re-computed.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The CPU speed is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Maximum Number of Concurrently Active Databases (numdb)

Configuration Type	Database manager
Applies to	

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type	Configurable	
Default [Range]		
	UNIX	8 [1 — 256]
	OS/2 and Windows NT Database server with local and remote clients	8 [1 — 256]
	OS/2 and Windows NT Database server with local clients and Satellite database server with local clients	3 [1 — 256]
Unit of Measure	Counter	

This parameter specifies the number of local databases that can be concurrently active (that is, have applications connected to them). In a partitioned database environment, it limits the number of active database partitions on a database partition server, whether that server is the coordinator node for the application or not.

Since each database takes up storage and an active database uses a new shared memory segment, you can reduce system resource usage by limiting the number of separate databases on your machine. However, arbitrarily reducing the number of databases is not the answer. That is, putting all data, no matter how unrelated, in one database will reduce disk space, but may not be a good idea. It is generally a good practice to only keep functionally related information in the same database.

Recommendation: It is generally best to set this value to the actual number of databases that are already defined to the database manager and to add a reasonable increment to account for future growth in the number of databases over the short term (for example, 6 months to 1 year). The actual increment should not be excessively large, but it should allow you to add new databases without having to frequently update this parameter.

Changing the *numdb* parameter may impact the total amount of memory allocated. As a result, frequent updates to this parameter are not recommended. When updating this parameter, you should consider the other configuration parameters that can allocate memory for a database or an application connected to that database, including:

- “Buffer Pool Size (buffpage)” on page 327
- “Maximum Storage for Lock List (locklist)” on page 335
- “Application Heap Size (applheapsz)” on page 345
- “Application Control Heap Size (app_ctl_heap_sz)” on page 340
- “Sort Heap Size (sortheap)” on page 342
- “Statement Heap Size (stmtheap)” on page 344
- “Application Support Layer Heap Size (aslheapsz)” on page 353
- “Database Heap (dbheap)” on page 329
- “Database System Monitor Heap Size (mon_heap_sz)” on page 358
- “Statistics Heap Size (stat_heap_sz)” on page 345

Transaction Processor Monitor Name (tp_mon_name)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default	No default
Valid Values	• CICS • MQ • ENCINA • CB • SF • TUXEDO • TOPEND • blank or some other value (for UNIX, OS/2, and Windows NT; no other possible values for Solaris or SINIX)

This parameter identifies the name of the transaction processing (TP) monitor product being used.

- If applications are run in a WebSphere Enterprise Edition CICS environment, this parameter should be set to “CICS”

- If applications are run in a WebSphere Enterprise Edition Encina environment, this parameter should be set to "ENCINA"
- If applications are run in a WebSphere Enterprise Edition Component Broker environment, this parameter should be set to "CB"
- If applications are run in an IBM MQSeries environment, this parameter should be set to "MQ"
- If applications are run in a BEA Tuxedo environment, this parameter should be set to "TUXEDO"
- If applications are run in an IBM San Francisco environment, this parameter should be set to "SF".

IBM WebSphere EJB and **Microsoft Transaction Server** users do not need to configure any value for this parameter.

If none of the above products are being used, this parameter should not be configured but left blank.

In previous versions of DB2 Universal Database in the OS/2 and Windows NT environments, this parameter contained the path and name of the DLL which contained the XA Transaction Manager's functions *ax_reg* and *ax_unreg*. This format is still supported. If the value of this parameter does not match any of the above TP Monitor names, it will be assumed that the value is a library name which contains the *ax_reg* and *ax_unreg* functions. This is true for UNIX, OS/2, and Windows NT environments.

TXSeries CICS and Encina Users: In previous versions of this product on OS/2 and Windows NT it was required to configure this parameter as "libEncServer:C" or "libEncServer:E". While this is still supported, it is no longer required. Configuring the parameter as "CICS" or "ENCINA" is sufficient.

MQSeries Users: In previous versions of this product on OS/2 and Windows NT it was required to configure this parameter as "mqmax". While this is still supported, it is no longer required. Configuring the parameter as "MQ" is sufficient.

Component Broker Users: In previous versions of this product on OS/2 and Windows NT it was required to configure this parameter as "somtrx1i". While this is still supported, it is no longer required. Configuring the parameter as "CB" is sufficient.

San Francisco Users: In previous versions of this product on OS/2 and Windows NT it was required to configure this parameter as "ibmsfDB2". While this is still supported, it is no longer required. Configuring the parameter as "SF" is sufficient.

The maximum length of the string that can be specified for this parameter is 19 characters.

It is also possible to configure this information in DB2 Universal Database's XA OPEN string. If multiple Transaction Processing Monitors are using a single DB2 instance, then it will be required to use this capability. Refer to the *Administration Guide: Planning* for additional information on using the XA OPEN string.

Machine Node Type (nodetype)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Informational

This parameter provides information about the DB2 products which you have installed on your machine and, as a result, information about the type of database managerconfiguration. The following are the possible values returned by this parameter and the products associated with that node type:
- **Database server with local and remote clients –** a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers.
- **Client –** a database client capable of accessing remote database servers.
- **Database server with local clients –** a DB2 relational database management system, supporting local database clients and capable of accessing other, remote database servers.
- **Partitioned database server with local and remote clients –** a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers, and capable of partition parallelism.
- **Satellite database server with local clients** a DB2 relational database management system, supporting local database clients and capable of accessing other, remote database servers.

For the numeric equivalents and API constants for these values, refer to the *Administrative API Reference*.

Default Charge-Back Account (dft_account_str)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	Null [any valid string]

With each application connect request, an accounting identifier consisting of a DB2 Connect-generated prefix and the user supplied suffix is sent from the application requester to a DRDA application server. This accounting information provides a mechanism for system administrators to associate resource usage with each user access.

Note: This parameter is only applicable to DB2 Connect.

The suffix is supplied by the application program calling the `sqlesact()` API or the user setting the environment variable DB2ACCOUNT. If a suffix is not supplied by either the API or environment variable, DB2 Connect uses the value of this parameter as the default suffix value. This parameter is particularly useful for down-level database clients (anything prior to version 2) that do not have the capability to forward an accounting string to DB2 Connect.

Recommendation: Set this accounting string using the following:
- Alphabetics (A through Z)
- Numerics (0 through 9)
- Underscore (_).

Java Development Kit 1.1 Installation Path (jdk11_path)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Client • Database server with local clients

- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type	Configurable
Default [Range]	Null [Valid path]
Related Parameters	• "Maximum Java Interpreter Heap Size (java_heap_sz)" on page 362

This parameter specifies the directory under which the Java Development Kit 1.1 is installed. The CLASSPATH and other environment variables used by the Java interpreter are computed from the value of this parameter.

Because there is no default for this parameter, you should specify a value for this parameter when you install the Java Development Kit.

Federated Database System Support (federated)

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	NO [YES; NO]

This parameter enables or disables support for applications submitting distributed requests for data managed by data sources (such as the DB2 Family and Oracle).

Instance Administration

The following parameters relate to security and administration of your database manager instance:

- "System Administration Authority Group Name (sysadm_group)" on page 458
- "System Control Authority Group Name (sysctrl_group)" on page 459
- "System Maintenance Authority Group Name (sysmaint_group)" on page 460
- "Authentication Type (authentication)" on page 461

- “Cataloging Allowed without Authority (catalog_noauth)” on page 462
- “Default Database Path (dftdbpath)” on page 463
- “LOGON Required for DB2START/DB2STOP (ss_logon)” on page 464
- “Trust All Clients (trust_allclnts)” on page 464
- “Trusted Clients Authentication (trust_clntauth)” on page 466

System Administration Authority Group Name (sysadm_group)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default	Null
Related Parameters	• “System Control Authority Group Name (sysctrl_group)” on page 459 • “System Maintenance Authority Group Name (sysmaint_group)” on page 460

System administration (SYSADM) authority is the highest level of authority within the database manager and controls all database objects. This parameter defines the group name with SYSADM authority for the database manager instance.

SYSADM authority is determined by the security facilities used in a specific operating environment.

- In the Windows 95 or Windows 98 operating system the SYSADM group must be NULL.

  This parameter must be “NULL” for Windows 95 or Windows 98 clients when system security is used because the Windows 95 or Windows 98 operating system does not store group information, thereby providing no way of determining if a user is a member of a designated SYSADM group. When a group name is specified, no user can be a member of it.
- For the Windows NT and Windows 2000 operating system, this parameter can be set to any local group that has a name of 8 characters or fewer, and

is defined in the Windows NT and Windows 2000 security database. If "NULL" is specified for this parameter, all members of the Administrators group have SYSADM authority.

- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the SYSADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the SYSADM group can be any valid UNIX group name.
- In OS/2, if the value specified for this parameter is "NULL", users defined as administrators in user profile management have SYSADM authority.

  If a group name is specified for this parameter, only users who belong to the group have SYSADM authority. The group specified can be any of the User Profile Management (UPM) groups. For more information on User Profile Management (UPM) groups, see your *DB2 for OS/2 Quick Beginnings* book.

If DCE security is used and *sysadm_group* is "NULL", the default DCE group name DB2ADMIN is used. A valid DCE principal whose authid mapping is DB2ADMIN must already exist. You can specify a different group name.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSADM_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

System Control Authority Group Name (sysctrl_group)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default	Null
Related Parameters	• "System Administration Authority Group Name (sysadm_group)" on page 458 • "System Maintenance Authority Group Name (sysmaint_group)" on page 460

This parameter defines the group name with system control (SYSCTRL) authority. SYSCTRL has privileges allowing operations affecting system resources, but does not allow direct access to data.

Attention: This parameter must be NULL for Windows 95 and Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 95 and Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSCTRL group. When a group name is specified, no user can be a member of it. This is not true when DCE authentication is used. In this situation, group names can be specified.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSCTRL_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

System Maintenance Authority Group Name (sysmaint_group)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default	Null
Related Parameters	• "System Administration Authority Group Name (sysadm_group)" on page 458 • "System Control Authority Group Name (sysctrl_group)" on page 459

This parameter defines the group name with system maintenance (SYSMAINT) authority. SYSMAINT has privileges to perform maintenance operations on all databases associated with an instance without having direct access to data.

Attention: This parameter must be NULL for Windows 95 and Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the

Windows 95 and Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSMAINT group. When a group name is specified, no user can be a member of it. This is not true when DCE authentication is used. In this situation, group names can be specified.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMAINT_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Authentication Type (authentication)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients • Satellite database server with local clients
Parameter Type	Configurable
Default [Range]	SERVER [CLIENT; SERVER; SERVER_ENCRYPT; DCS; DCS_ENCRYPT; DCE; DCE_SERVER_ENCRYPT; KERBEROS; KRB_SERVER_ENCRYPT]

This parameter determines how and where authentication of a user takes place.

If authentication is SERVER, then the user ID and password are sent from the client to the server so authentication can take place on the server. The value SERVER_ENCRYPT provides the same behavior as SERVER, except that any passwords sent over the network are encrypted.

A value of CLIENT indicates that all authentication takes place at the client, so no authentication needs to be performed at the server.

A value of DCS indicates that authentication takes place at the host or AS/400 system. The value DCS_ENCRYPT provides the same behavior as DCS, except that any passwords sent over the network are encrypted. If you are using APPC and a communications product that does not expose the client's password to the DB2 server, you can specify DCS to obtain:

- SERVER-type authentication for non-DRDA clients
- CLIENT-type authentication for DRDA clients

A value of DCE means that authentication is performed at the DCE server using DCE Security Services. The value DCE_SERVER_ENCRYPT provides the same behavior as DCE, except any passwords sent over the network are encrypted. The DCE_SERVER_ENCRYPT value is for use on a server only. This value indicates that the server can accept either DCE authentication or SERVER_ENCRYPT authentication.

A value of KERBEROS means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication. With an authentication type of KRB_SERVER_ENCRYPT at the server and clients that support the Kerberos security system, then the effective system authentication type is KERBEROS. If the clients do not support the Kerberos security system, then the effective system authentication type is equivalent to SERVER_ENCRYPT.

Note: The Kerberos authentication types are only supported on servers running Windows 2000.

Authentication values that support password encryption include: SERVER_ENCRYPT, DCS_ENCRYPT, DCE_SERVER_ENCRYPT, and KRB_SERVER_ENCRYPT. These values provide the same function as SERVER, DCS, DCE, and KERBEROS respectively in terms of authentication location, except that any passwords that flow are encrypted at the source and require decryption at the target, as specified by the authentication type cataloged at the source. Encrypted and non-encrypted values with matching authentication locations can then be used to choose different encryption combinations between the client and gateway or the gateway and server, without affecting where authentication occurs.

For the numeric equivalents and API constants for these values, refer to the *Administrative API Reference*.

For more information on when and why to use DCE or DCS, and authentication issues related to federated databases, refer to the “Controlling Database Access” chapter in *Administration Guide: Implementation*.

Recommendation: Typically, the default (SERVER) is adequate. If you have incoming requests that are handled by Kerberos, DB2 Connect, or DCE, refer to the “Controlling Database Access” chapter in *Administration Guide: Implementation*.

Cataloging Allowed without Authority (catalog_noauth)

Configuration Type	Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type
Configurable

Default [Range]

Database server with local and remote clients; Database server with local and remote clients
NO [NO (0) — YES (1)]

Client; Database server with local clients; Satellite database server with local clients
YES [NO (0) — YES (1)]

This parameter specifies whether users are able to catalog and uncatalog databases and nodes, or DCS and ODBC directories, without SYSADM authority. The default value (0) for this parameter indicates that SYSADM authority is required. When this parameter is set to 1 (yes), SYSADM authority is not required.

Default Database Path (dftdbpath)

Configuration Type
Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type
Configurable

Default [Range]

UNIX
Home directory of instance owner [any existing path]

OS/2 and Windows NT
Drive on which DB2 is installed [any existing path]

This parameter contains the default file path used to create databases under the database manager. If no path is specified when a database is created, the database is created under the path specified by the *dftdbpath* parameter.

In a partitioned database environment, you should ensure that the path on which the database is being created is not an NFS-mounted path (on UNIX-based platforms), or a network drive (in the Windows NT environment). The specified path must physically exist on each database partition server. To avoid confusion, it is best to specify a path that is locally mounted on each database partition server. The maximum length of the path is 205 characters. The system appends the node name to the end of the path.

Given that databases can grow to a large size and that many users could be creating databases (depending on your environment and intentions), it is often convenient to be able to have all databases created and stored in a specified location. It is also good to be able to isolate databases from other applications and data both for integrity reasons and for ease of backup and recovery.

For UNIX-based environments, the length of the *dftdbpath* name cannot exceed 215 characters and must be a valid, absolute, path name. For OS/2 and Windows NT, the *dftdbpath* can be a drive letter, optionally followed by a colon.

Recommendation: If possible, put high volume databases on a different disk than other frequently accessed data, such as the operating system files and the database logs.

LOGON Required for DB2START/DB2STOP (ss_logon)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients
Parameter Type	Configurable
Default [Range]	YES [NO (0), YES (1)]

This parameter is applicable to the OS/2 environment only. By accepting the default for this parameter, a LOGON user ID and password is required before issuing a DB2START or DB2STOP.

Trust All Clients (trust_allclnts)

Configuration Type	Database manager
Applies to	

	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	YES [NO, YES, DRDAONLY]
Related Parameters	• "Authentication Type (authentication)" on page 461 • "Trusted Clients Authentication (trust_clntauth)" on page 466

This parameter is only active when the *authentication* parameter is set to CLIENT.

This parameter and *trust_clntauth* are used to determine where users are validated to the database environment.

By accepting the default of "YES" for this parameter, all clients are treated as trusted clients. This means that the server assumes that a level of security is available at the client and the possibility that users can be validated at the client.

This parameter can only be changed to "NO" if the *authentication* parameter is set to CLIENT. If this parameter is set to "NO", the untrusted clients must provide a userid and password combination when they connect to the server. Untrusted clients are operating system platforms that do not have a security subsystem for authenticating users.

Setting this parameter to "DRDAONLY" protects against all clients except DRDA clients from DB2 for MVS and OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

When *trust_allclnts* is set to "DRDAONLY", the *trust_clntauth* parameter is used to determine where the clients are authenticated. If *trust_clntauth* is set to "CLIENT", authentication occurs at the client. If *trust_clntauth* is set to "SERVER", authentication occurs at the client if no password is provided, and at the server if a password is provided.

Refer to "Selecting an Authentication Method for Your Server" in the *Administration Guide: Implementation* for more information on trusted clients.

Trusted Clients Authentication (trust_clntauth)

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	CLIENT [CLIENT, SERVER]
Related Parameters	• “Authentication Type (authentication)” on page 461 • “Trust All Clients (trust_allclnts)” on page 464

This parameter specifies whether a trusted client is authenticated at the server or the client when the client provides a userid and password combination for a connection. This parameter (and *trust_allclnts*) is only active if the *authentication* parameter is set to CLIENT. If a user ID and password are not provided, the client is assumed to have validated the user, and no further validation is performed at the server.

If this parameter is set to “CLIENT” (the default), the trusted client can connect without providing a userid and password combination, and the assumption is that the operating system has already authenticated the user. If it is set to “SERVER”, the user ID and password will be validated at the server.

The numeric value for CLIENT is 0. The numeric value for SERVER is 1.

Refer to “Selecting an Authentication Method for Your Server” in the *Administration Guide: Implementation* for more information on trusted clients.

Part 4. Appendixes

Appendix A. DB2 Registry and Environment Variables

The following is a list of DB2 registry variables and environment variables that you may need to know about to get up and running. Each has a brief description; some may not apply to your environment.

You can view a list of all supported registry variables by using:

```
db2set -lr
```

You can change the value for a variable for the current session by using:

```
db2set registry_variable_name=new_value
```

To update environment variables, the `set` command must be used and then the system rebooted.

The values for the changed registry variables must be set before the DB2START command is issued. Refer to *Administration Guide: Planning* for more information on changing and using registry variables.

Table 21. General Registry Variables

Variable Name	Operating System	Values
Description		
DB2ACCOUNT	All	Default=null
The accounting string that is sent to the remote host. Refer to the *DB2 Connect User's Guide* for details.		
DB2BIDI	All	Default=NO Values: YES or NO
This variable enables bidirectional support and the *db2codepage* variable is used to declare the code page to be used. Refer to the *Administration Guide: Planning* in the National Language Support appendix for additional information on bidirectional support.		
DB2CODEPAGE	All	Default: derived from the language ID, as specified by the operating system.
Specifies the code page of the data presented to DB2 for database client application. The user should not set *db2codepage* unless explicitly stated in DB2 documents, or asked to do so by DB2 service. Setting *db2codepage* to a value not supported by the operating system can produce unexpected results. Normally, you do not need to set *db2codepage* because DB2 automatically derives the code page information from the operating system.		
DB2COUNTRY	All	Default: derived from the language ID, as specified by the operating system.
Specifies the country code of the client application, which influences date and time formats.		

Table 21. General Registry Variables (continued)

Variable Name / Description	Operating System	Values
DB2DBDFT	All	Default=null
Specifies the database alias name of the database to be used for implicit connects. If an application has no database connection but SQL statements are issued, an implicit connect will be made provided the DB2DBDFT environment variable has been defined with a default database.		
DB2DBMSADDR	Windows 32-bit operating systems	Default= 0x20000000 for Windows NT, 0x90000000 for Windows 95 Value: 0x20000000 to 0xB0000000 in increments of 0x10000
Specifies the default database manager shared memory address in hexadecimal format. If *db2start* fails due to a shared memory address collision, this registry variable can be modified to force the database manager instance to allocate its shared memory at a different address.		
DB2_DISABLE_FLUSH_LOG	All	Default= OFF Value: ON or OFF
Specifies if you wish to disable the inclusion of the last active log file in any on-line backups. When an on-line backup completes, the last active log file is truncated, closed, and archived as part of the backup. This provides you with a complete back up including all of the logs required for the restoring of that backup. You may wish to disable the inclusion of the last active log file if you are concerned that you are wasting portions of the Log Sequence Number (LSN) space. Each time an active log file is truncated, the LSN is incremented by an amount proportional to the space truncated. If you perform a very large number of on-line backups each day, you may wish to disable the inclusion of the last active log file. You may also wish to disable the inclusion of the last active log file if you find you are receiving log full messages a short time after the completion of the on-line backup. When a log file is truncated, the reserved active log space is incremented by the amount proportional to the size of the truncated log. The active log space is freed once the truncated log file is reclaimed. The reclamation occurs a short time after the log file becomes inactive. It is during the short interval in-between that you may receive log full messages.		

DB2DISCOVERYTIME	OS/2 and Windows 32-bit operating systems	Default=40 seconds,
		
		Minimum=20 seconds

Specifies the amount of time that SEARCH discovery will search for DB2 systems.		
DB2INCLUDE	All	Default=current directory
Specifies a path to be used during the processing of the SQL INCLUDE text-file statement during DB2 PREP processing. It provides a list of directories where the INCLUDE file might be found. Refer to the *Application Development Guide* for descriptions of how *db2include* is used in the different precompiled languages.		

Table 21. General Registry Variables (continued)

Variable Name / Description	Operating System	Values
DB2INSTDEF	OS/2 and Windows 32-bit operating systems	Default=DB2
Sets the value to be used if *DB2INSTANCE* is not defined.		
DB2INSTOWNER	Windows NT	Default=null
The registry variable created in the DB2 profile registry when the instance is first created. This variable is set to the name of the instance-owning machine.		
DB2_LIC_STAT_SIZE	All	Default=null Range: 0 to 32767
The registry variable is used to determine the maximum size (in MBs) of the file containing the license statistics for the system. A value of zero turns the license statistic gathering off. If the variable is not recognized or not defined, the variable defaults to unlimited. The statistics are displayed using the license center.		

DB2NBDISCOVERRCVBUFS	All	Default=16 buffers,
		
		Minimum=16 buffers

This variable is used for NetBIOS search discovery. The variable specifies the number of concurrent discovery responses that can be received by a client. If the client receives more concurrent responses than are specified by this variable, then the excess responses are discarded by the NetBIOS layer. The default is sixteen (16) NetBIOS receive buffers. If a number less than the default value is chosen, then the default is used.		
DB2OPTIONS	All except Windows 3.1 and Macintosh	Default=null
Sets command line processor options.		
DB2SLOGON	Windows 3.x	Default=null, Values: YES or NO
Enables a secure logon in DB2 for Windows 3.x. If *db2slogon*=`YES` DB2 does not write user IDs and passwords to a file, but instead uses a segment of memory to maintain them. When *db2slogon* is enabled, the user must logon each time Windows 3.x is started.		
DB2TIMEOUT	Windows 3.x and Macintosh	Default=(not set)
Used to control the timeout period for Windows 3.x and Macintosh clients during long SQL queries. After the timeout period has expired a dialog box pops up asking if the query should be interrupted or allowed to continue. The minimum value for this variable is 30 seconds. If *db2timeout* is set to a value between 1 and 30, the default minimum value will be used. If *db2timeout* is set to a value of 0, or a negative value, the timeout feature is disabled. This feature is disabled by default.		

Table 21. General Registry Variables (continued)

Variable Name / Description	Operating System	Values
DB2TRACENAME	Windows 3.x and Macintosh	Default= DB2WIN.TRC (on Windows 3.x), DB2MAC.TRC (on Macintosh)
On Windows 3.x and Macintosh, specifies the name of the file where trace information is stored. The default for each system is saved in your current instance directory (for example, `\sqllib\db2`). We strongly recommend that you specify the full path name when naming the trace file.		
DB2TRACEON	Windows 3.x and Macintosh	Default=NO Values: YES or NO
On Windows 3.x and Macintosh, turns trace on to provide information to IBM in case of a problem. (It is not recommended that you turn trace on unless you encounter a problem you cannot resolve.) Refer to the *Troubleshooting Guide* for information on using the trace facility with clients.		
DB2TRCFLUSH	Windows 3.x and Macintosh	Default=NO Values: YES or NO
On Windows 3.x and Macintosh, *db2trcflush* can be used in conjunction with *db2traceon*=`YES`. Setting *db2trcflush*=`YES` will cause each trace record to be written immediately into the trace file. This will slow down your DB2 system considerably, so the default setting is *db2trcflush*=`NO`. This setting is useful in cases where an application hangs the system and requires the system to be rebooted. Setting this keyword guarantees that the trace file and trace entries are not lost by the reboot.		
DB2TRCSYSERR	Windows 3.x and Macintosh	Default=1 Values: 1-32767
Specifies the number of system errors to trace before the client turns off tracing. The default value traces one system error, after which, trace is turned off.		
DB2YIELD	Windows 3.x	Default=NO Values: YES or NO
Specifies the behavior of the Windows 3.x client while communicating with a remote server. When set to `NO`, the client will not yield the CPU to other Windows 3.x applications, and the Windows environment is halted while the client application is communicating with the remote server. You must wait for the communications operation to complete before you can resume any other tasks. When set to `YES`, your system functions as normal. It is recommended that you try to run your application with *db2yield*=`YES`. If your system crashes, you will need to set *db2yield*=`NO`. For application development, ensure your application is written to accept and handle Windows messages while waiting for a communications operation to complete.		

Table 22. System Environment Variables

Variable Name Description	Operating System	Values
DB2CONNECT_IN_APP_PROCESS	All	Default=YES Values: YES or NO
When setting this variable to NO, local DB2 Connect clients on a DB2 Connect Enterprise Edition machine are forced to run within an agent. Some advantages of running within an agent are that local clients are able to be monitored and that they can use SYSPLEX support.		
DB2DOMAINLIST	Windows NT server only	Default=null Values: A list of Windows NT domain names separated by commas (“,”).
This variable is effective only when CLIENT authentication is set in the Database Manager configuration; and is needed if a single signon from Windows NT desktop is required in a Windows NT domain environment. The list defines the domains which the userID will be authenticated against. **Note:** When this variable is set together with CLIENT authentication specified, only connection requests from DB2 Universal Database for Windows NT clients Version 7 (or later) are allowed.		
DB2ENVLIST	UNIX	Default: null
Lists specific variable names for either stored procedures or user-defined functions. By default, the **db2start** command filters out all user environment variables except those prefixed with **DB2** or **db2**. If specific registry variables must be passed to either stored procedures or user-defined functions, you can list the variable names in the *db2envlist* registry variable. Separate each variable name by one or more spaces. DB2 constructs its own PATH and LIBPATH, so if PATH or LIBPATH is specified in *db2envlist*, the actual value of the variable name is appended to the end of the DB2-constructed value.		
DB2INSTANCE	All	Default=*db2instdef* on OS/2 and Windows 32-bit operating systems.
The environment variable used to specify the instance that is active by default. On UNIX, users must specify a value for *DB2INSTANCE*.		
DB2INSTPROF	OS/2, Windows 3.x, and Windows 32-bit operating systems	Default: null
The environment variable used to specify the location of the instance directory on OS/2, Windows 3.x, and Windows 32-bit operating systems, if different than *DB2PATH*.		
DB2LIBPATH	UNIX	Default: null
Specifies the value or LIBPATH in the *db2libpath* registry variable. The value of LIBPATH cannot be inherited between parent and child processes if the user ID has changed. Since the **db2start** executable is owned by root, DB2 cannot inherit the LIBPATH settings of end users. If you list the variable name, LIBPATH, in the *db2envlist* registry variable, you must also specify the value of LIBPATH in the *db2libpath* registry value. The **db2start** executable then reads the value of *db2libpath* and append this value to the end of the DB2-constructed LIBPATH.		

Table 22. System Environment Variables (continued)

Variable Name	Operating System	Values
Description		
DB2NODE	All	Default: null Values: 1 to 999
Used to specify the target logical node of a DB2 Extended Enterprise Edition database partition server that you want to attach to or connect to. If this variable is not set, the target logical node defaults to the logical node which is defined with port `0` on the machine.		
DB2_PARALLEL_IO	All	Default: null Values: * (meaning every table space) or a comma-separated list of more than one defined table space
While reading or writing data from and to table space containers, DB2 may use parallel I/O if the number of containers in the database is greater than one. To force parallel I/O for a single container, use this registry variable. After setting the registry variable, issue a DB2STOP and then enter DB2START to allow the changes to take effect.		
DB2PATH	OS/2, Windows 3.x, and Windows 32-bit operating systems	Default: (varies by operating system)
The environment variable used to specify the directory where the product is installed on OS/2, Windows 3.x, and Windows 32-bit operating systems.		
DB2_STRIPED_CONTAINERS	All	Default: null Values: ON, null
When using RAID devices for table space containers, it is suggested that the table space be created with an extent size that is equal to, or a multiple of, the RAID stripe size. However, because of the one-page container tag, the extents will not line up with the RAID stripes, and it may be necessary during an I/O request to access more physical disks than would be optimal. When using DMS table space containers this problem is avoided by allocating the tag its own (full) extent. This avoids the problem but does require an extra extent of overhead within the container. After setting this registry variable, issue a DB2STOP and then enter DB2START to allow the changes to take effect.		

Table 23. Communications Variables

Variable Name	Operating System	Values
Description		
DB2CHECKCLIENTINTERVAL	AIX, server only	Default=0 Values: A numeric value greater than zero.

Table 23. Communications Variables (continued)

Variable Name	Operating System	Values
Description		
Used to verify the status of APPC client connections. Permits early detection of client termination, rather than waiting until after the completion of the query. When set to zero, no check will be made. When set to a numerical value greater than zero, the value represents DB2 internal work units. For guidance, the following check frequency values are given: Low frequency use 300; medium frequency use 100; high frequency use 50. Checking more frequently for client status while executing a database request lengthens the time taken to complete the queries. If the DB2 workload is heavy (that is, it involves many internal requests), then setting DB2CHECKCLIENTINTERVAL to a low value has a greater impact on performance than in a situation where the workload is light and most of the time DB2 is waiting.		
DB2COMM	All, server only	Default=null Values: any combination of APPC, IPXSPX, NETBIOS, NPIPE, TCPIP
Specifies the communication managers that are started when the database manager is started. If this is not set, no DB2 communications managers are started at the server.		
DB2_FORCE_NLS_CACHE	AIX, HP_UX, Solaris	Default=FALSE Values: TRUE or FALSE
Used to eliminate the chance of lock contention in multi-threaded applications. When this registry variable is "TRUE", the code page and country code information is saved the first time a thread accesses it. From that point, the cached information is used for any other thread that requests this information. This eliminates lock contention and results in a performance benefit in certain situations. This setting should not be used if the application changes locale settings between connections. It is likely not needed in such a situation anyway, since multi-threaded applications typically do not change their locale settings because it is not "thread-safe" to do so.		
DB2NBADAPTERS	OS/2 and Windows NT	Default=0 Range: 0-15, Multiple values should be separated by commas
Used to specify which local adapters to use for DB2 NetBIOS LAN communications. Each local adapter is specified using its logical adapter number.		
DB2NBCHECKUPTIME	OS/2 and Windows NT, server only	Default=1 minute Values: 1-720
Specifies the time interval between each invocation of the NetBIOS protocol checkup procedure. Checkup time is specified in minutes. Lower values will ensure that the NetBIOS protocol checkup runs more often, freeing up memory and other system resources left when unexpected agent/session termination occurs.		

Table 23. Communications Variables (continued)

Variable Name / Description	Operating System	Values
DB2NBINTRLISTENS	OS/2 and Windows NT, server only	Default=1 Values: 1-10 Multiple values should be separated by commas
Specifies the number of NetBIOS listen send commands (NCBs) that will be asynchronously issued in readiness for remote client interrupts. This flexibility is provided for ″interrupt active″ environments to ensure that interrupt calls from remote clients will be able to establish connections when servers are busy servicing other remote interrupts. Setting *db2nbintrlistens* to a lower value will conserve NetBIOS sessions and NCBs at the server. However, in an environment where client interrupts are common, you may need to set *db2nbintrlistens* to a higher value in order to be responsive to interrupting clients. **Note:** Values specified are position sensitive; they relate to the corresponding value positions for *db2nbadapters*.		
DB2NBRECVBUFFSIZE	OS/2 and Windows NT, server only	Default=4096 bytes Range: 4096-65536
Specifies the size of the DB2 NetBIOS protocol receive buffers. These buffers are assigned to the NetBIOS receive NCBs. Lower values conserve server memory, while higher values may be required when client data transfers are larger.		
DB2NBBRECVNCBS	OS/2 and Windows NT, server only	Default=10 Range: 1-99
Specifies the number of NetBIOS ″receive_any″ commands (NCBs) that the server will issue and maintain during operation. This value may be adjusted depending on the number of remote clients to which your server is connected. Lower values will conserve server resources. **Note:** Each adapter in use can have its own unique receive NCB value specified by *db2nbbrecvncbs*. The values specified are position sensitive; they relate to the corresponding value positions for *db2nbadapters*.		
DB2NBRESOURCES	OS/2 and Windows NT server only	Default=null
Specifies the number of NetBIOS resources to allocate for DB2 use in a multi-context environment. This variable is restricted to multi-context client operation.		
DB2NBSENDNCBS	OS/2 and Windows NT, server only	Default=6 Range: 1-720

Table 23. Communications Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies the number of send NetBIOS commands (NCBs) that the server will reserve for use. This value may be adjusted depending on the number of remote clients your server is connected to. Setting *db2nbsendncbs* to a lower value will conserve server resources. However, you may need to set it to a higher value to prevent the server from waiting to send to a remote client when all other send commands are in use.		
DB2NBSESSIONS	OS/2 and Windows NT, server only	Default=null Range: 5-254
Specifies the number of sessions that DB2 should request to be reserved for DB2 use. The value of *db2nbsessions* can be set to request a specific session for each adapter specified using *db2nbadapters*. **Note:** Values specified are position sensitive; they relate to the corresponding value positions for *db2nbadapters*.		
DB2NBXTRANCBS	OS/2 and Windows NT, server only	Default=5 per adapter Range: 5-254
Specifies the number of "extra" NetBIOS commands (NCBs) the server will need to reserve when the **db2start** command is issued. The value of *db2nbxtrancbs* can be set to request a specific session for each adapter specified using *db2nbadapters*.		
DB2NETREQ	Windows 3.x	Default=3 Range: 0-25
Specifies the number of NetBIOS requests that can be run concurrently on Windows 3.x clients. The higher you set this value, the more memory below the 1MB level will be used. When the concurrent number of requests to use NetBIOS services reaches the number you have set, subsequent incoming requests for NetBIOS services are held in a queue and become active as the current requests complete. If you enter 0 (zero) for *db2netreq*, the Windows database client issues NetBIOS calls in synchronous mode using the NetBIOS wait option. In this mode, the database client allows only the current NetBIOS request to be active and does not process another one until the current request has completed. This can affect other application programs. The 0 value is provided for backwards compatibility only. It is strongly recommended that 0 not be used.		
DB2RETRY	OS/2 and Windows NT	Default=0 Range: 0-20 000
The number of times DB2 attempts to restart the APPC listener. If the SNA subsystem at the server/gateway is down, this profile variable, in conjunction with *db2retrytime*, can be used to automatically restart the APPC listener without disrupting client communications using other protocols. In such a scenario, it is no longer necessary to stop and restart DB2 to reinstate the APPC client communications.		
DB2RETRYTIME	OS/2 and Windows NT	Default=1 minute Range: 0-7 200 minutes

Table 23. Communications Variables (continued)

Variable Name / Description	Operating System	Values
In increments of one minute, the number of minutes that DB2 allows between performing successive retries to start the APPC listener. If the SNA subsystem at the server/gateway is down, this profile variable, in conjunction with *db2retry*, can be used to automatically restart the APPC listener without disrupting client communications using other protocols. In such a scenario, it is no longer necessary to stop and restart DB2 to reinstate the APPC client communications.		
DB2SERVICETPINSTANCE	OS/2 and Windows NT	Default=null
Used to support incoming APPC connections from DB2 workstation V.1 clients or from the DB2 MVS database. When the **db2start** command is invoked, the instance specified will start the APPC listeners for the following TP names: • DB2INTERRUPT • x'07'68 • x'07'6SN		
DB2SOSNDBUF	Windows 95 and Windows NT	Default=32767
Specifies the value of TCP/IP send buffers on Windows 95 and Windows NT operating systems.		
DB2SYSPLEX_SERVER	OS/2, Windows NT, and UNIX	Default=null
Specifies whether SYSPLEX exploitation when connected to DB2 for OS/390 is enabled. If this registry variable is not set (which is the default), or is set to a non-zero value, exploitation is enabled. If this registry variable is set to zero (0), exploitation is disabled. When set to zero, SYSPLEX exploitation is disabled for the gateway regardless of how the DCS database catalog entry has been specified. For more information see the *Command Reference* and the **CATALOG DCS DATABASE** command.		
DB2TCPCONNMGRS	All	Default=1 on serial machines; square root of the number of processors rounded up to a maximum of eight connection managers on symmetric multiprocessor machines. Values: 1 to 8

Table 23. Communications Variables (continued)

Variable Name / **Description**	**Operating System**	**Values**
The default number of connection managers is created if the registry variable is not set. If the registry variable is set, the value assigned here overrides the default value. The number of TCP/IP connection managers specifed up to a maximum of 8 is created. If less than one is specified then DB2TCPCONNMGRS is set to a value of one and a warning is logged that the value is out of range. If greater than eight is specified then DB2TCPCONNMGRS is set to a value of eight and a warning is logged that the value is out of range. Values between one and eight are used as given. When there is greater than one connection manager created, connection throughput should improve when multiple client connections are received simultaneously. There may be additional TCP/IP connection manager process (on UNIX) or threads (on OS/2 and Windows operating systems) if the user is running on a SMP machine, or has modified the DB2TCPCONNMGRS registry variable. Additional processes or threads require additional storage.		
DB2_VI_ENABLE	Windows NT	Default=OFF Values: ON or OFF
Specifies whether to use the Virtual Interface Architecture (VIA) communication protocol or not. If this registry variable is “ON”, then FCM will use VI for inter-node communication. If this registry variable is “OFF”, then FCM will use TCP/IP for inter-node communication. **Note:** The value of this registry variable must be the same across all the database partitions in the instance.		
DB2_VI_VIPL	Windows NT	Default= `vipl.dll`
Specifies the name of the Virtual Interface Provider Library (VIPL) that will be used by DB2. In order to load the library successfully, the library name used in this registry variable must be in the *PATH* user environment variable. The currenly supported implementations all use the same library name.		
DB2_VI_DEVICE	Windows NT	Default=null Values: `nic0`
Specifies the symbolic name of the device or Virtual Interface Provider Instance associated with the Network Interface Card (NIC). Independent hardware vendors (IHVs) each produce their own NIC. Only one (1) NIC is allowed per Windows NT machine; Multiple logical nodes on the same physical machine will share the same NIC. The currenly supported implementations all use the same symbolic name.		

Table 24. DCE Directory Variables

Variable Name / **Description**	**Operating System**	**Values**
DB2DIRPATHNAME	OS/2, UNIX, and Windows 32-bit operating systems	Default=null

Table 24. DCE Directory Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies a temporary override of the DIR_PATH_NAME parameter value in the database manager configuration file. If a directory server is used and the target of a CONNECT statement or ATTACH command is not explicitly cataloged, then the target is concatenated with DB2DIRPATHNAME (if specified) to form the fully qualified DCE name. **Note:** The *db2dirpathname* variable has no effect on the instance's global name, which is always identified by the database manager configuration parameters DIR_PATH_NAME and DIR_OBJ_NAME.		
DB2CLIENTADPT	OS/2 and Windows 32-bit operating systems	Default=null Range: 0-15
Specifies the client adapter number for NETBIOS protocol on OS/2 and Windows 32-bit operating systems. The *db2clientadpt* value overrides the DFT_CLIENT_ADPT parameter value in the database manager configuration file.		
DB2CLIENTCOMM	OS/2, UNIX, and Windows 32-bit operating systems	Default=null
Specifies a temporary override of the DFT_CLIENT_COMM parameter value in the database manager configuration file. If both DFT_CLIENT_COMM and *db2clientcomm* are not specified, then the first protocol found in the object is used. If either one or both of them are specified, then only the first matching protocol will be used. In either case, no retry is attempted if the first connect fails.		
DB2ROUTE	OS/2, UNIX, and Windows 32-bit operating systems	Default=null
Specifies the name of the Routing Information Object the client uses when it connects to a database with a different database protocol. The *db2route* value overrides the ROUTE_OBJ_NAME parameter value in the database manager configuration file.		

Table 25. Command Line Variables

Variable Name	Operating System	Values
Description		
DB2BQTIME	All	Default=1 second Maximum value: 1 second
Specifies the amount of time the command line processor front end will sleep before checking if the back end process is active and establishing a connection to it.		
DB2BQTRY	All	Default=60 retries Minimum value: 0 retries
Specifies the number of times the command line processor front end process tries to determine whether the back end process is already active. It works in conjunction with *db2bqtime*.		

Table 25. Command Line Variables (continued)

Variable Name / Description	Operating System	Values
DB2IQTIME	All	Default=5 seconds Minimum value: 1 second
Specifies the amount of time the command line processor back end process waits on the input queue for the front end process to pass commands.		
DB2RQTIME	All	Default=5 seconds Minimum value: 1 second
Specifies the amount of time the command line processor back end process waits for a request from the front end process.		

Table 26. MPP Configuration Variables

Variable Name	Operating System	Values
DB2ATLD_PORTS	DB2 UDB EEE on AIX, Solaris, and Windows NT	Default= 6000:6063 Value: num1:num2 where both are between 1 and 65535, and num1<=num2
Specifies the range of port numbers used for the AutoLoader utility's internal TCPIP communication. If not set, AutoLoader uses the internal default port range 6000:6063. When you have other applications using the AutoLoader default port range, this variable can be used to select an alternate port range.		
DB2ATLD_PWFILE	DB2 UDB EEE on AIX, Solaris, and Windows NT	Default=null Value: a file path expression
Specifies a path to a file that contains a password used during AutoLoader authentication. If not set, AutoLoader either extracts the password from its configuration file or prompts you interactively. Using this variable will address password security concerns and allows the separation of AutoLoader configuration information from authentication information.		
DB2CHGPWD_EEE	DB2 UDB EEE on AIX and Windows NT	Default=null Values: YES or NO
Specifies whether you are allowing other users to change passwords on AIX or Windows NT EEE systems. You must ensure that the passwords for all partitions or nodes are maintained centrally using either a Windows NT domain controller on Windows NT, or NIS on AIX. If not maintained centrally, passwords may not be consistent across all partitions or nodes. This could result in a password only being changed at the database partition to which the user connects to make the change. In order to modify this global registry variable, you must be at the root directory and on the DAS instance.		
DB2_FORCE_FCM_BP	AIX	Default=NO Values: YES or NO

Table 26. MPP Configuration Variables (continued)

Variable Name	Operating System	Values
This registry variable is applicable to DB2 UDB EEE for AIX when using multiple logical partitions. When DB2START is issued, DB2 allocates the FCM buffers from the database global memory or, if there is not enough room there, from a separate shared memory segment which is used by all FCM daemons (for that instance) on the same physical machine. Which it chooses is largely dependent on the number of FCM buffers to be created (which, in turn, is determined by the FCM_NUM_BUFFERS database manager configuration parameter). If this registry variable is set to YES, the FCM buffers are always created in a separate memory segment. When the FCM buffers are created in a separate memory segment, the communication between FCM daemons of different logical partitions on the same physical node occurs through shared memory. Otherwise, FCM daemons on the same node communicate through UNIX Sockets. The advantage of communicating through shared memory in this way is that it is faster. The disadvantage is that there is one fewer shared memory segments available for other uses, most notably database buffer pools. This reduces the maximum size of database buffer pools.		
DB2_NUM_FAILOVER_NODES	DB2 UDB on AIX, Solaris, and Windows NT	Default: 2 Values: 0 to the number of logical nodes
Specifies the number of nodes that can be used as failover nodes in a high availability environment. With high availability, if a node fails, then the node can be restarted as a second logical node on a different host. The number used with this variable determines how much memory is reserved for FCM resources for failover nodes. For example, host A has two logical nodes: 1 and 2; and host B has two logical nodes: 3 and 4. Assume DB2_NUM_FAILOVER_NODES is set to 2. During DB2START, both host A and host B will reserve enough memory for FCM so that up to four logical nodes could be managed. Then if one host fails, the logical nodes for the failing host could be restarted on the other host.		
DB2PORTRANGE	Windows NT	Values: nnnn:nnnn
This value is set to the TCP/IP port range used by FCM so that any additional partitions created on another machine will also have the same port range.		

Table 27. SQL Compiler Variables

Variable Name	Operating System	Values
Description		
DB2_ANTIJOIN	All	Default=NO Values: YES or NO
This registry variable is applicable to DB2 Universal Database EEE environments. If specifies whether the optimizer will search for opportunities to transform “NOT EXISTS” subqueries into anti-joins which can be processed more efficiently by DB2.		
DB2_CORRELATED_PREDICATES	All	Default=OFF Values: ON or OFF

Table 27. SQL Compiler Variables (continued)

Variable Name Description	Operating System	Values
When there are unique indexes on correlated columns in a join, and this registry variable is ON, the optimizer attempts to detect and compensate for correlation of join predicates. When this registry variable is ON, the optimizer uses the KEYCARD information of unique index statistics to detect cases of correlation, and dynamically adjusts the combined selectivities of the correlated predicates, thus obtaining a more accurate estimate of the join size and cost.		
DB2_HASH_JOIN	All	Default=NO Values: YES or NO
Specifies hash join as a possible join method when compiling an access plan.		
DB2_LIKE_VARCHAR	All	Default=NO Values: YES, NO, or a floating point constant between 0 and 6.001
Specifies how the optimizer works with a predicate of the form `COLUMN LIKE '%XXXXXX%'` where the `xxxxxx` is any string of characters. For all predicates, the optimizer has to estimate how many rows match the predicate. For LIKE predicates with leading and trailing `%` characters, the optimizer assumes that the COLUMN being matched has a structure of a series of elements concatenated together to form the entire column. The optimizer then estimates the length of each element based on the length of the string enclosed in the `%` characters.		
DB2_NEW_CORR_SQ_FF	All	Default=OFF Values: ON or OFF

Affects the selectivity value computed by the SQL optimizer for certain subquery predicates when it is set to "ON". If can be used to improve the accuracy of the selectivity value of equality subquery predicates that use the MIN or MAX aggregate function in the SELECT list of the subquery. For example:
`SELECT * FROM T WHERE`
`T.COL = (SELECT MIN(T.COL)`
`FROM T WHERE ...)`		

DB2_PRED_FACTORIZE	All	Default=NO Value: YES or NO

Table 27. SQL Compiler Variables (continued)

Variable Name	Operating System	Values
Description		

Specifies whether the optimizer will search for opportunities to extract additional predicates from disjuncts. In some circumstances, the additional predicates can alter the estimated cardinality of the intermediate and final result sets. With the following query:

```
SELECT n1.empno,
       n1.lastname
  FROM employee n1,
       employee n2
  WHERE
   ((n1.lastname='SMITH'
  AND n2.lastname='JONES')
  OR (n1.lastname='JONES'
  AND n2.lastname='SMITH'))
```

the optimizer can generate the following additional predicates:

```
SELECT n1.empno,
       n1.lastname
  FROM employee n1,
       employee n2
  WHERE n1.lastname IN
   ('SMITH', 'JONES')
  AND n2.lastname IN
   ('SMITH', 'JONES')
  AND
   ((n1.lastname='SMITH'
  AND n2.lastname='JONES')
  OR (n1.lastname='JONES'
  AND n2.lastname='SMITH'))
```

Table 28. Performance Variables

Variable Name	Operating System	Values
Description		
DB2_AVOID_PREFETCH	All	Default=OFF, Values: ON or OFF
Specifies whether or not prefetch should be used during crash recovery. If *db2_avoid_prefetch*=`ON`, prefetch is not used.		
DB2_BINSORT	AIX	Default=YES Values: YES or NO

Table 28. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
Enables a new sort algorithm that reduces the CPU time and elapsed time of sorts. This new algorithm extends the extremely efficient integer sorting technique of DB2 UDB to all sort datatypes such as BIGINT, CHAR, VARCHAR, FLOAT, and DECIMAL, as well as combinations of these datatypes. To enable this new algorithm, use the following command: `db2set DB2_BINSORT = yes`		
DB2BPVARS	Windows NT	Default=path

Specifies the path to a file containing parameters used when tuning buffer pools. The currently supported parameters are: NT_SCATTER_DMSFILE, NT_SCATTER_DMSDEVICE, and NT_SCATTER_SMS.
 	
For each of these parameters, the default is zero (or OFF); and the possible values include: zero (or OFF) and 1 (or ON). Each parameter is used to turn scatter read on for the respective type of containers. Each can only be enabled (turned ON) if DB2NTNOCACHE is set to ON in the registry. A warning message is written to the db2diag.log if DB2NTNOCACHE is set to OFF (or not set), and scatter read remains disabled. The parameters are recommended for systems with a large amount of sequential prefetching against the respective type of containers and you have already decided to use DB2NTNOCACHE set to OFF.
 	
An example of how to set the path to the file is shown:
`db2set DB2BPVARS =`
`   f:\BPVARSFILE`
 	
The content of the file is any of these parameters in the form:
`parameter=value`		

DB2CHKPTR	All	Default=OFF, Values: ON or OFF
Specifies whether or not pointer checking for input is required.		
DB2_DARI_LOOKUP_ALL	All	Default=OFF Values: ON or OFF
Specifies whether or not the UDB server will perform a catalog lookup for ALL DARIs and stored procedures before looking in the *function* subdirectory of the *sqllib* subdirectory; and in the *unfenced* subdirectory of the *function* subdirectory of the *sqllib* subdirectory. **Note:** For stored procedures of PARAMETER TYPE DB2DARI that are located in the directories mentioned above, setting this value to “ON” will degrade performance since the catalog lookup will be performed possibly on another node in an EEE configuration) before the function directories are searched.		

Table 28. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
DB2_EXTENDED_OPTIMIZATION	All	Default=OFF Values: ON or OFF
Specifies whether or not the query optimizer uses optimization extensions to improve query performance. The extensions may not improve query performance in all environments. Testing should be done to determine individual query performance improvements.		
DB2MEMDISCLAIM	AIX	Default=null Values: YES or NO
Depending on the workload being executed and the pool agents configuration, you may run into a situation where the committed memory for each DB2 agent will stay above 32 MB even when the agent is idle. This behavior is expected and usually results in good performance as the memory is kept for fast re-use. However, on a memory constrained system, this may not be a desirable side effect. To avoid this condition, issue the following: `db2set DB2MEMDISCLAIM = yes` Disclaiming memory tells the AIX operating system to stop paging the area so that it no longer occupies any real storage. Setting DB2MEMDISCLAIM to “YES” tells DB2 UDB to disclaim some or all memory once freed, depending on the value given with DB2MEMMAXFREE. If DB2MEMMAXFREE is null, then all of the memory is disclaimed once freed. If DB2MEMMAXFREE is given a value, then only some of the memory is disclaimed once freed (up to the value given in DB2MEMMAXFREE). This ensures that the memory is made readily available for other processes as soon as it is freed. See also DB2MEMMAXFREE. These two registry variables work together.		
DB2MEMMAXFREE	AIX	Default=null Values: 4000000 to 256000000

Specifies the amount of free memory that is retained by each DB2 agent. You may set this variable to a value between 4 and 256 MB. We recommend that if you use this feature, you specify a value of 8 MB:
`db2set DB2MEMMAXFREE`
`   = 8000000`
 	
See also DB2MEMDISCLAIM. These two registry variables work together.		

DB2_MMAP_READ	AIX	Default=ON , Values: ON or OFF
Used in conjunction with *db2_mmap_write* to allow DB2 to use mmap as an alternate method of I/O. In most environments, mmap should be used to avoid operating system locks when multiple processes are writing to different sections of the same file. However, perhaps you migrated from Parallel Edition V1.2 where the default was OFF allowing AIX chaching of DB2 data read from JFS filesystems into memory (outside the buffer pool). If you want the comparable performance with DB2 UDB, you can either increase the size of the buffer pool, or change *db2_mmap_read* and *db2_mmap_write* to `OFF`.		

Table 28. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
DB2_MMAP_WRITE	AIX	Default=ON Values: ON or OFF
Used in conjunction with *db2_mmap_read* to allow DB2 to use mmap as an alternate method of I/O. In most environments, mmap should be used to avoid operating system locks when multiple processes are writing to different sections of the same file. However, perhaps you migrated from Parallel Edition V1.2 where the default was OFF allowing AIX caching of DB2 data read from JFS filesystems into memory (outside the buffer pool). If you want the comparable performance with DB2 UDB, you can either increase the size of the buffer pool, or change *db2_mmap_read* and *db2_mmap_write* to `OFF`.		
DB2_NO_PKG_LOCK	All	Default=OFF Values: ON or OFF
Allows the Global SQL Cache to operate without the use of package locks to protect cached package entries. (Package locks are internal system locks.) To improve performance (because acquiring and freeing locks takes time), you can now choose to work in a "no package lock" mode. In this mode, certain database operations are not allowed. These operations may include: operations that invalidate packages, operations that inoperate packages, and operations that directly change a package.		
DB2NTMEMSIZE	Windows NT	Default=(varies by memory segment)

Windows NT requires that all shared memory segments be reserved at DLL initialization time in order to guarantee matching addresses across processes. *DB2NTMEMSIZE* has been introduced to permit the user to override the DB2 defaults on Windows NT if necessary. In most situations, the default values should be sufficient. The memory segments, default sizes, and override options are: 1) Database Kernel: default size is 16777216 (16 MB); override option is DBMS:<number of bytes>. 2) Parallel FCM Buffers: default size is 22020096 (21 MB); override option is FCM:<number of bytes>. 3) Database Admin GUI: default size is 33554432 (32 MB); override option is DBAT:<number of bytes>. 4) Fenced Stored Procedures: default size is 16777216 (16 MB); override option is APLD:<number of bytes>. More than one segment may be overridden by separating the override options with a semi-colon (;). For example, to limit the database kernel to approximately 256K, and the FCM buffers to approximately 64 MB, use:
`db2set DB2NTMEMSIZE=`
`DBMS:256000;FCM:64000000`		

DB2NTNOCACHE	Windows NT	Default=OFF Value: ON or OFF
Specifies whether or not DB2 will open database files with a NOCACHE option. If *db2ntnocache*=ON, file system caching is eliminated. If *db2ntnocache*=OFF, the operating system caches DB2 files. This applies to all data except for files that contain LONG FIELDS or LOBS. Eliminating system caching allows more memory to be available to the database so that the buffer pool or sortheap can be increased.		

Table 28. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
DB2NTPRICLASS	Windows NT	Default=null Value: R, H, (any other value)
Sets the priority class for the DB2 instance (program DB2SYSCS.EXE). There are three priority classes: • NORMAL_PRIORITY_CLASS (the default priority class) • REALTIME_PRIORITY_CLASS (set by using "R") • HIGH_PRIORITY_CLASS (set by using "H") This variable is used in conjunction with individual thread priorities (set using *DB2PRIORITIES*) to determine the absolute priority of DB2 threads relative to other threads in the system. **Note:** Care should be taken when using this variable. Misuse could adversely affect overall system performance. For more information, please refer to the *SetPriorityClass()* API in the Win32 documentation.		
DB2NTWORKSET	Windows NT	Default=1,1
Used to modify the minimum and maximum working set size available to DB2. By default, when Windows NT is not in a paging situation, a process's working set can grow as large as needed. However, when paging occurs, the maximum working set that a process can have is approximately 1 MB. DB2NTWORKSET allows you to override this default behavior. Specify DB2NTWORKSET for DB2 using the syntax *db2ntworkset*=`min,max`, where min and max are expressed in megabytes.		
DB2_OVERRIDE_BPF	All	Default=not set Values: a positive numeric number of pages
Specifies the size of the buffer pool, in pages, to be created at database activation, or first connection, time. It is useful when failures occur during database activation or first connection resulting from memory constraints. Should even a minimal buffer pool of 16 pages not be brought up by the database manager, then the user can try again after specifying a smaller number of pages using this environment variable. The memory constraint could arise either because of a real memory shortage (which is rare); or, because of the attempt by the database manager to allocate large, inaccurately configured buffer pools. This value, if set, will override the current buffer pool size.		
DB2PRIORITIES	All	Values setting is platform dependent.
Controls the priorities of DB2 processes and threads.		
DB2_RR_TO_RS	All	Default=NO Values: YES or NO

Table 28. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
When set to YES, the RR isolation level is, effectively, downgraded to RS for user tables. RR semantics are no longer provided in the database manager instance. If your applications do not require RR semantics, this registry variable can be used to reduce the next-key lock contention problems that can sometimes occur under RR.		
DB2_SORT_AFTER_TQ	All	Default=NO Values: YES or NO
Specifies how the optimizer works with directed table queues in a partitioned database when the receiving end requires the data to be sorted, and the number of receiving nodes is equal to the number of sending nodes. When *DB2_SORT_AFTER_TQ*= `NO`, the optimizer tends to sort at the sending end, and merge the rows at the receiving end. When *DB2_SORT_AFTER_TQ*= `YES`, the optimizer tends to transmit the rows unsorted, not merge at the receiving end, and sort the rows at the receiving end after receiving all the rows.		

Table 29. Data Links Variables

Variable Name **Description**	Operating System	Values
DLFM_BACKUP_DIR_NAME	AIX, Windows NT	Default: null Values: TSM or any valid path
Specifies the backup device to use. If you change the setting of this registry variable between TSM and a path at run-time, the archived files are not moved. Only new backups are place in the new location. Previously archived files are not moved.		
DLFM_BACKUP_LOCAL_MP	AIX, Windows NT	Default: null Values: any valid path to the local mount point in the DFS system
Specifies the fully qualified path to a mount point in the DFS system. When a path is given, it is used instead of the path given with DLFM_BACKUP_DIR_NAME.		
DLFM_BACKUP_TARGET	AIX, Windows NT	Default: null Values: LOCAL, TSM, XBSA
Specifies the type of backup used.		
DLFM_BACKUP_TARGET_LIBRARY	AIX, Windows NT	Default: null Values: any valid path to the DLL or shared library name

Table 29. Data Links Variables (continued)

Variable Name	**Operating System**	**Values**
Description		
Specifies the fully qualified path to the DLL or shared library. This library is loaded using the *libdfmxbsa.a* library.		
DLFM_ENABLE_STPROC	AIX, Windows NT	Default: NO Values: YES or NO
Specifies whether a stored procedure is used to link groups of files.		
DLFM_FS_ENVIRONMENT	AIX, Windows NT	Default: NATIVE Values: NATIVE or DFS
Specifies the environment in which Data Links servers operate. NATIVE indicates that the Data Links server is in a single machine situation where the server can take over files on its own machine. DFS indicates that the Data Links server is in a distributed filesystem (DFS) environment where the server can take over files throughout the filesystem. Mixing DFS filesets and native filesystems is not allowed.		
DLFM_GC_MODE	AIX, Windows NT	Default: PASSIVE Values: SLEEP, PASSIVE, or ACTIVE
Specifies the control of garbage file collection on the Data Links server. When set to SLEEP, no garbage collection occurs. When set to PASSIVE, garbage collection runs only if no other transactions are running. When set to ACTIVE, garbage collection runs even if other transactions are running.		
DLFM_INSTALL_PATH	AIX, Windows NT	Default On AIX: /usr/lpp/ db2_06_00 /adm On NT: DB2PATH /bin Range: any valid path
Specifies the path where the data links executables are installed.		
DLFM_LOG_LEVEL	AIX, Windows NT	Default: LOG_INFO Values: LOG_CRIT, LOG_DEBUG, LOG_ERR, LOG_INFO, LOG_NOTICE, LOG_WARNING
Specifies the level of diagnostic information to be recorded.		
DLFM_PORT	All except Windows 3.n	Default: 50100 Values: any valid port number
Specifies the port number used to communicate with the Data Links servers running the DB2 Data Links Manager. This environment variable is only used when a table contains a “DATALINKS” column.		

Table 30. Miscellaneous Variables

Variable Name / Description	Operating System	Values
DB2ADMINSERVER	OS/2, Windows 95, Windows NT, and UNIX	Default=null
Specifies which DB2 instance is set up as the DB2 Administration Server.		
DB2CLIINIPATH	All	Default=null
Used to override the default path of the DB2 CLI/ODBC configuration file (db2cli.ini) and specify a different location on the client. The value specified must be a valid path on the client system.		
DB2DEFPREP	All	Default=NO Values: ALL, YES, or NO
Simulates the runtime behavior of the DEFERRED_PREPARE precompile option for applications that were precompiled prior to this option becoming available. For example, if a DB2 v2.1.1 or earlier application were run in a DB2 v2.1.2 or later environment, *db2defprep* could be used to indicate the desired 'deferred prepare' behavior.		
DB2_DJ_COMM	All	Default=null Values include: libdrda.a, libsqlnet.a, libnet8.a, libdrda.dll, libsqlnet.dll, libnet8.dll, and so on.
Specifies the wrapper libraries that are loaded when the database manager is started. Specifying this variable reduces the run-time cost of loading frequently used wrappers. Other values for other operating systems are supported (the .dll extension is for the Windows NT operating system; the .a extension is for the AIX operating system). Library names vary by protocol and operating system. This variable is not available unless the database manager parameter *federated* is set to YES.		
DB2DMNBCKCTLR	Windows NT	Default=null Values: ? or a domain name
If you know the name of the domain for which DB2 server is the backup domain controller, set *db2dmnbckctlr*=`DOMAIN_NAME`. The DOMAIN_NAME must be in upper case. To have DB2 determine the domain for which the local machine is a backup domain controller, set *db2dmnbckctlr*=?. If the *db2dmnbckctlr* profile variable is not set or is set to blank, DB2 performs authentication at the primary domain controller. **Note:** DB2 does not use an existing backup domain controller by default because a backup domain controller can get out of synchronization with the primary domain controller, causing a security exposure. Getting out of synchronization can occur when the primary domain controller's security database is updated but the changes are not propagated to a backup domain controller. This could occur if there are network latencies or if the computer browser service is not operational.		
DB2_ENABLE_LDAP	All	Default=NO Values: YES or NO

Table 30. Miscellaneous Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies whether or not the Lightweight Directory Access Protocol (LDAP) is used. LDAP is an access method to directory services.		
DB2_FALLBACK	Windows NT	Default=OFF Values: ON or OFF
This variable allows you to force all database connections off during the fallback processing. It is used in conjunction with the failover support in the Windows NT environment with Microsoft Cluster Server (MSCS). If *DB2_FALLBACK* is not set or is set to OFF, and a database connection exists during the fall back, the DB2 resource cannot be brought offline. This will mean the fallback processing will fail.		
DB2_FORCE_TRUNCATION	All	Default=NO Values: YES or NO
Used during restart recovery. If set to "NO", it will halt restart recovery if it is determined that a bad page is stopping the restart recovery too soon (that is, all active logs have not been read). This is usually caused by a bad page in one of the logs. The user can set this variable to "YES" to signal restart recovery that it should continue processing as if the end of logs was reached. After setting the variable to "YES", logs not read during restart recovery are overwritten when the database becomes active again. The default is "NO", which is **not** to proceed if a bad page is not found. Use this variable only under the direction from IBM Service personnel.		
DB2_GRP_LOOKUP	Windows NT	Default=null Values: LOCAL, DOMAIN
This variable is used to tell DB2 where to validate user accounts and perform group member lookup. Set the variable to LOCAL to force DB2 to always enumerate groups and validate user accounts on the DB2 server. Set the variable to DOMAIN to force DB2 to always enumerate groups and validate user accounts on the Windows NT domain to which the user account belongs.		
DB2LDAP_BASEDN	All	Default=null Values: Any valid base domain name.
Specifies the base domain name for the LDAP directory.		
DB2LDAPCACHE	All	Default=YES Values: YES or NO

Specifies that the LDAP cache is to be enabled. This cache is used to catalog the database, node, and DCS directories on the local machine.
 	
To ensure that you have the latest entries in the cache, do the following:
 	
`REFRESH LDAP DB DIR`
`REFRESH LDAP NODE DIR`
 	
These commands update and remove incorrect entries from the database directory and the node directory.		

Table 30. Miscellaneous Variables (continued)

Variable Name / Description	Operating System	Values
DB2LDAP_CLIENT_PROVIDER	Windows 95/98/NT/2000 only	Default=null (Microsoft, if available, is used; otherwise IBM is used.) Values: IBM or Microsoft
When running in a Windows environment, DB2 supports using either Microsoft LDAP clients or IBM LDAP clients to access the LDAP directory. This registry variable is used to explicitly select the LDAP client to be used by DB2. **Note:** To display the current value of this registry variable, use the `db2set` command: `db2set DB2LDAP_CLIENT_PROVIDER`		
DB2LDAPHOST	All	Default=null Values: Any valid hostname.
Specifies the hostname of the location for the LDAP directory.		
DB2LDAP_SEARCH_SCOPE	All	Default= DOMAIN Values: LOCAL, DOMAIN, GLOBAL
Specifies the search scope for information found in partitions or domains in the Lightweight Directory Access Protocol (LDAP). “LOCAL” disables searching in the LDAP directory. “DOMAIN” only searches in LDAP for the current directory partition. “GLOBAL” searches in LDAP in all directory partitions until the object is found.		
DB2LOADREC	All	Default=null
Used to override the location of the load copy during roll forward. If the user has changed the physical location of the load copy, *db2loadrec* must be set before issuing the roll forward.		
DB2LOCK_TO_RB	All	Default=null Values: Statement
Specifies whether lock timeouts cause the entire transaction to be rolled-back, or only the current statement. If *db2lock_to_rb* is set to `STATEMENT`, locked timeouts cause only the current statement is rolled back. Any other setting results in transaction rollback.		
DB2NOEXITLIST	All	Default=OFF Values: ON or OFF

Table 30. Miscellaneous Variables (continued)

Variable Name	**Operating System**	**Values**
Description		
If defined, this variable indicates to DB2 not to install an exit list handler in applications and not to perform a COMMIT. Normally, DB2 installs a process exit list handler in applications and the exit list handler performs a COMMIT operation if the application ends normally. For applications that dynamically load the DB2 library and unload it before the application terminates, the invocation of the exit list handler fails because the handler routine is no longer loaded in the application. If your application operates in this way, you should set the *DB2NOEXITLIST* variable and ensure your application explicitly invokes all required COMMITs.		
DB2REMOTEPREG	Windows 95 and Windows NT	Default=null Value: Any valid Windows 95 or Windows NT machine name
Specifies the remote machine name that contains the Win32 registry list of DB2 instance profiles and DB2 instances. The value for DB2REMOTEPREG should only be set once after DB2 is installed, and should not be modified. Use this variable with extreme caution.		
DB2ROUTINE_DEBUG	AIX and Windows NT	Default=OFF Values: ON, OFF
Specifies whether to enable the debug capability for Java stored procedures. If you are not debugging Java stored procedures, use the default, OFF. There is a performance impact to enable debugging. Refer to *Application Development Guide* for more information about debugging Java stored procdures.		
DB2SORCVBUF	Windows 95 and Windows NT	Default=32767
Specifies the value of TCP/IP receive buffers on Windows 95 and Windows NT operating systems.		
DB2SORT	All, server only	Default=null
Specifies the location of a library to be loaded at runtime by the LOAD utility. The library contains the entry point for functions used in sorting indexing data. Use *db2sort* to exploit vendor-supplied sorting products for use with the LOAD utility in generating table indexes. The path supplied must be relative to the database server.		
DB2SYSTEM	Windows NT, Windows 95, OS/2, and UNIX	Default=null

Table 30. Miscellaneous Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies the name that is used by your users and database administrators to identify the DB2 server system. If possible, this name should be unique within your network. This name is displayed in the system level of the Control Center's object tree to aid administrators in the identification of server systems that can be administered from the Control Center. When using the 'Search the Network' function of the Client Configuration Assistant, DB2 discovery returns this name and it is displayed at the system level in the resulting object tree. This name aids users in identifying the system that contains the database they wish to access. A value for *db2system* is set at installation time as follows: • On Windows NT, or Windows 95, the setup program sets it equal to the computer name specified for the Windows system. • On OS/2, the user is prompted to enter the DB2SYSTEM name during the installation process. • On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.		
DB2UPMPR	OS/2	Default=ON Values: ON or OFF
Specifies whether or not the UPM logon screen will display on the screen when the user enters the wrong user ID or password on OS/2.		

Appendix B. Explain Tables and Definitions

The Explain tables capture access plans when the Explain facility is activated. The following Explain tables and definitions are described in this section:

- "EXPLAIN_ARGUMENT Table" on page 498
- "EXPLAIN_INSTANCE Table" on page 502
- "EXPLAIN_OBJECT Table" on page 504
- "EXPLAIN_OPERATOR Table" on page 506
- "EXPLAIN_PREDICATE Table" on page 508
- "EXPLAIN_STATEMENT Table" on page 511
- "EXPLAIN_STREAM Table" on page 513
- "ADVISE_INDEX Table" on page 515
- "ADVISE_WORKLOAD Table" on page 518

The Explain tables must be created before Explain can be invoked. To create them, use the sample command line processor input script provided in the EXPLAIN.DDL file located in the 'misc' subdirectory of the 'sqllib' directory. Connect to the database where the Explain tables are required. Then issue the command: `db2 -tf EXPLAIN.DDL` and the tables will be created. See "Table Definitions for Explain Tables" on page 518 for more information.

The population of the Explain tables by the Explain facility will neither activate any triggers nor activate any referential or check constraints. For example, if an insert trigger were defined on the EXPLAIN_INSTANCE table and an eligible statement were explained, the trigger would not be activated.

See "Chapter 7. SQL Explain Facility" on page 201 for more details on the Explain facility.

Legend for the Explain Tables:

Heading	**Explanation**
Column name	Name of the column
Data Type	Data type of the column
Nullable?	Yes: Nulls are permitted No: Nulls are not permitted
Key?	PK: Column is part of a primary key FK: Column is part of a foreign key
Description	Description of the column

EXPLAIN_ARGUMENT Table

The EXPLAIN_ARGUMENT table represents the unique characteristics for each individual operator, if there are any.

Table 31. EXPLAIN_ARGUMENT Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	INTEGER	No	No	Unique ID for this operator within this query.
ARGUMENT_TYPE	CHAR(8)	No	No	The type of argument for this operator.
ARGUMENT_VALUE	VARCHAR(1024)	Yes	No	The value of the argument for this operator. NULL if the value is in LONG_ARGUMENT_VALUE.
LONG_ARGUMENT_VALUE	CLOB(1M)	Yes	No	The value of the argument for this operator, when the text will not fit in ARGUMENT_VALUE. NULL if the value is in ARGUMENT_VALUE.

Table 32. ARGUMENT_TYPE and ARGUMENT_VALUE Column Values

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
AGGMODE	COMPLETE PARTIAL INTERMEDIATE FINAL	Partial aggregation indicators.
BITFLTR	TRUE FALSE	Hash Join will use a bit filter to enhance performance.
CSETEMP	TRUE FALSE	Temporary Table over Common Subexpression Flag.
DIRECT	TRUE	Direct fetch indicator.
DUPLWARN	TRUE FALSE	Duplicates Warning flag.

Table 32. ARGUMENT_TYPE and ARGUMENT_VALUE Column Values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
EARLYOUT	TRUE FALSE	Early out indicator.
ENVVAR	Each row of this type will contain: • Environment variable name • Environment variable value	Environment variable affecting the optimizer
FETCHMAX	IGNORE INTEGER	Override value for MAXPAGES argument on FETCH operator.
GROUPBYC	TRUE FALSE	Whether Group By columns were provided.
GROUPBYN	Integer	Number of comparison columns.
GROUPBYR	Each row of this type will contain: • Ordinal value of column in group by clause (followed by a colon and a space) • Name of Column	Group By requirement.
INNERCOL	Each row of this type will contain: • Ordinal value of column in order (followed by a colon and a space) • Name of Column • Order Value **(A)** Ascending **(D)** Descending	Inner order columns.
ISCANMAX	IGNORE INTEGER	Override value for MAXPAGES argument on ISCAN operator.
JN_INPUT	INNER OUTER	Indicates if operator is the operator feeding the inner or outer of a join.
LISTENER	TRUE FALSE	Listener Table Queue indicator.
MAXPAGES	ALL NONE INTEGER	Maximum pages expected for Prefetch.
MAXRIDS	NONE INTEGER	Maximum Row Identifiers to be included in each list prefetch request.
NUMROWS	INTEGER	Number of rows expected to be sorted.
ONEFETCH	TRUE FALSE	One Fetch indicator.

Table 32. ARGUMENT_TYPE and ARGUMENT_VALUE Column Values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
OUTERCOL	Each row of this type will contain: • Ordinal value of column in order (followed by a colon and a space) • Name of Column • Order Value **(A)** Ascending **(D)** Descending	Outer order columns.
OUTERJN	LEFT RIGHT	Outer join indicator.
PARTCOLS	Name of Column	Partitioning columns for operator.
PREFETCH	LIST NONE SEQUENTIAL	Type of Prefetch Eligible.
RMTQTEXT	Query text	Remote Query Text
ROWLOCK	EXCLUSIVE NONE REUSE SHARE SHORT (INSTANT) SHARE UPDATE	Row Lock Intent.
ROWWIDTH	INTEGER	Width of row to be sorted.
SCANDIR	FORWARD REVERSE	Scan Direction.
SCANGRAN	INTEGER	Intra-partition parallelism, granularity of the intra-partition parallel scan, expressed in SCANUNITs.
SCANTYPE	LOCAL PARALLEL	intra-partition parallelism, Index or Table scan.
SCANUNIT	ROW PAGE	Intra-partition parallelism, scan granularity unit.
SERVER	Remote server	Remote server
SHARED	TRUE	Intra-partition parallelism, shared TEMP indicator.
SLOWMAT	TRUE FALSE	Slow Materialization flag.
SNGLPROD	TRUE FALSE	Intra-partition parallelism sort or temp produced by a single agent.

Table 32. ARGUMENT_TYPE and ARGUMENT_VALUE Column Values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
SORTKEY	Each row of this type will contain: • Ordinal value of column in key (followed by a colon and a space) • Name of Column • Order Value **(A)** Ascending **(D)** Descending	Sort key columns.
SORTTYPE	PARTITIONED SHARED ROUND ROBIN REPLICATED	Intra-partition parallelism, sort type.
TABLOCK	EXCLUSIVE INTENT EXCLUSIVE INTENT NONE INTENT SHARE REUSE SHARE SHARE INTENT EXCLUSIVE SUPER EXCLUSIVE UPDATE	Table Lock Intent.
TQDEGREE	INTEGER	intra-partition parallelism, number of subagents accessing Table Queue.
TQMERGE	TRUE FALSE	Merging (sorted) Table Queue indicator.
TQREAD	READ AHEAD STEPPING SUBQUERY STEPPING	Table Queue reading property.
TQSEND	BROADCAST DIRECTED SCATTER SUBQUERY DIRECTED	Table Queue send property.
TQTYPE	LOCAL	Intra-partition parallelism, Table Queue.
TRUNCSRT	TRUE	Truncated sort (limits number of rows produced).
UNIQUE	TRUE FALSE	Uniqueness indicator.
UNIQKEY	Each row of this type will contain: • Ordinal value of column in key (followed by a colon and a space) • Name of Column	Unique key columns.
VOLATILE	TRUE	Volatile table

EXPLAIN_INSTANCE Table

The EXPLAIN_INSTANCE table is the main control table for all Explain information. Each row of data in the Explain tables is explicitly linked to one unique row in this table. The EXPLAIN_INSTANCE table gives basic information about the source of the SQL statements being explained as well as information about the environment in which the explanation took place.

For the definition of this table, see "EXPLAIN_INSTANCE Table Definition" on page 521.

Table 33. EXPLAIN_INSTANCE Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	PK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	PK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	PK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	PK	Schema, or qualifier, of source of Explain request.
EXPLAIN_OPTION	CHAR(1)	No	No	Indicates what Explain Information was requested for this request. Possible values are: **P** PLAN SELECTION
SNAPSHOT_TAKEN	CHAR(1)	No	No	Indicates whether an Explain Snapshot was taken for this request. Possible values are: **Y** Yes, an Explain Snapshot(s) was taken and stored in the EXPLAIN_STATEMENT table. Regular Explain information was also captured. **N** No Explain Snapshot was taken. Regular Explain information was captured. **O** Only an Explain Snapshot was taken. Regular Explain information was not captured.
DB2_VERSION	CHAR(7)	No	No	Product release number for DB2 Universal Database which processed this explain request. Format is vv.rr.m, where: **vv** Version Number **rr** Release Number **m** Maintenance Release Number

Table 33. EXPLAIN_INSTANCE Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
SQL_TYPE	CHAR(1)	No	No	Indicates whether the Explain Instance was for static or dynamic SQL. Possible values are: **S** Static SQL **D** Dynamic SQL
QUERYOPT	INTEGER	No	No	Indicates the query optimization class used by the SQL Compiler at the time of the Explain invocation. The value indicates what level of query optimization was performed by the SQL Compiler for the SQL statements being explained.
BLOCK	CHAR(1)	No	No	Indicates what type of cursor blocking was used when compiling the SQL statements. For more information, see the BLOCK column in SYSCAT.PACKAGES. Possible values are: **N** No Blocking **U** Block Unambiguous Cursors **B** Block All Cursors
ISOLATION	CHAR(2)	No	No	Indicates what type of isolation was used when compiling the SQL statements. For more information, see the ISOLATION column in SYSCAT.PACKAGES. Possible values are: **RR** Repeatable Read **RS** Read Stability **CS** Cursor Stability **UR** Uncommitted Read
BUFFPAGE	INTEGER	No	No	Contains the value of the BUFFPAGE database configuration setting at the time of the Explain invocation.
AVG_APPLS	INTEGER	No	No	Contains the value of the AVG_APPLS configuration parameter at the time of the Explain invocation.
SORTHEAP	INTEGER	No	No	Contains the value of the SORTHEAP database configuration setting at the time of the Explain invocation.
LOCKLIST	INTEGER	No	No	Contains the value of the LOCKLIST database configuration setting at the time of the Explain invocation.
MAXLOCKS	SMALLINT	No	No	Contains the value of the MAXLOCKS database configuration setting at the time of the Explain invocation.

Table 33. EXPLAIN_INSTANCE Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
LOCKS_AVAIL	INTEGER	No	No	Contains the number of locks assumed to be available by the optimizer for each user. (Derived from LOCKLIST and MAXLOCKS.)
CPU_SPEED	DOUBLE	No	No	Contains the value of the CPUSPEED database manager configuration setting at the time of the Explain invocation.
REMARKS	VARCHAR(254)	Yes	No	User-provided comment.
DBHEAP	INTEGER	No	No	Contains the value of the DBHEAP database configuration setting at the time of Explain invocation.
COMM_SPEED	DOUBLE	No	No	Contains the value of the COMM_BANDWIDTH database configuration setting at the time of Explain invocation.
PARALLELISM	CHAR(2)	No	No	Possible values are: • N = No parallelism • P = Intra-partition parallelism • IP = Inter-partition parallelism • BP = Intra-partition parallelism and inter-partition parallelism
DATAJOINER	CHAR(1)	No	No	Possible values are: • N = Non-federated systems plan • Y = Federated systems plan

EXPLAIN_OBJECT Table

The EXPLAIN_OBJECT table identifies those data objects required by the access plan generated to satisfy the SQL statement.

Table 34. EXPLAIN_OBJECT Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.

Table 34. EXPLAIN_OBJECT Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
OBJECT_SCHEMA	VARCHAR(128)	No	No	Schema to which this object belongs.
OBJECT_NAME	VARCHAR(128)	No	No	Name of the object.
OBJECT_TYPE	CHAR(2)	No	No	Descriptive label for the type of object.
CREATE_TIME	TIMESTAMP	Yes	No	Time of Object's creation; null if a table function.
STATISTICS_TIME	TIMESTAMP	Yes	No	Last time of update to statistics for this object; null if statistics do not exist for this object.
COLUMN_COUNT	SMALLINT	No	No	Number of columns in this object.
ROW_COUNT	INTEGER	No	No	Estimated number of rows in this object.
WIDTH	INTEGER	No	No	The average width of the object in bytes. Set to -1 for an index.
PAGES	INTEGER	No	No	Estimated number of pages that the object occupies in the buffer pool. Set to -1 for a table function.
DISTINCT	CHAR(1)	No	No	Indicates if the rows in the object are distinct (i.e. no duplicates) Possible values are: **Y** Yes **N** No
TABLESPACE_NAME	VARCHAR(128)	Yes	No	Name of the table space in which this object is stored; set to null if no table space is involved.
OVERHEAD	DOUBLE	No	No	Total estimated overhead, in milliseconds, for a single random I/O to the specified table space. Includes controller overhead, disk seek, and latency times. Set to -1 if no table space is involved.
TRANSFER_RATE	DOUBLE	No	No	Estimated time to read a data page, in milliseconds, from the specified table space. Set to -1 if no table space is involved.
PREFETCHSIZE	INTEGER	No	No	Number of data pages to be read when prefetch is performed. Set to -1 for a table function.
EXTENTSIZE	INTEGER	No	No	Size of extent, in data pages. This many pages are written to one container in the table space before switching to the next container. Set to -1 for a table function.
CLUSTER	DOUBLE	No	No	Degree of data clustering with the index. If >= 1, this is the CLUSTERRATIO. If >= 0 and < 1, this is the CLUSTERFACTOR. Set to -1 for a table, table function, or if this statistic is not available.

Table 34. EXPLAIN_OBJECT Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
NLEAF	INTEGER	No	No	Number of leaf pages this index object's values occupy. Set to -1 for a table, table function, or if this statistic is not available.
NLEVELS	INTEGER	No	No	Number of index levels in this index object's tree. Set to -1 for a table, table function, or if this statistic is not available.
FULLKEYCARD	BIGINT	No	No	Number of distinct full key values contained in this index object. Set to -1 for a table, table function, or if this statistic is not available.
OVERFLOW	INTEGER	No	No	Total number of overflow records in the table. Set to -1 for an index, table function, or if this statistic is not available.
FIRSTKEYCARD	BIGINT	No	No	Number of distinct first key values. Set to −1 for a table, table function or if this statistic is not available.
FIRST2KEYCARD	BIGINT	No	No	Number of distinct first key values using the first {2,3,4} columns of the index. Set to −1 for a table, table function or if this statistic is not available.
FIRST3KEYCARD	BIGINT	No	No	
FIRST4KEYCARD	BIGINT	No	No	
SEQUENTIAL_PAGES	INTEGER	No	No	Number of leaf pages located on disk in index key order with few or no large gaps between them. Set to −1 for a table, table function or if this statistic is not available.
DENSITY	INTEGER	No	No	Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percentage (integer between 0 and 100). Set to −1 for a table, table function or if this statistic is not available.

Table 35. Possible OBJECT_TYPE Values

Value	Description
IX	Index
TA	Table
TF	Table Function

EXPLAIN_OPERATOR Table

The EXPLAIN_OPERATOR table contains all the operators needed to satisfy the SQL statement by the SQL compiler.

Table 36. EXPLAIN_OPERATOR Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.

Table 36. EXPLAIN_OPERATOR Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	INTEGER	No	No	Unique ID for this operator within this query.
OPERATOR_TYPE	CHAR(6)	No	No	Descriptive label for the type of operator.
TOTAL_COST	DOUBLE	No	No	Estimated cumulative total cost (in timerons) of executing the chosen access plan up to and including this operator.
IO_COST	DOUBLE	No	No	Estimated cumulative I/O cost (in data page I/Os) of executing the chosen access plan up to and including this operator.
CPU_COST	DOUBLE	No	No	Estimated cumulative CPU cost (in instructions) of executing the chosen access plan up to and including this operator.
FIRST_ROW_COST	DOUBLE	No	No	Estimated cumulative cost (in timerons) of fetching the first row for the access plan up to and including this operator. This value includes any initial overhead required.
RE_TOTAL_COST	DOUBLE	No	No	Estimated cumulative cost (in timerons) of fetching the next row for the chosen access plan up to and including this operator.
RE_IO_COST	DOUBLE	No	No	Estimated cumulative I/O cost (in data page I/Os) of fetching the next row for the chosen access plan up to and including this operator.
RE_CPU_COST	DOUBLE	No	No	Estimated cumulative CPU cost (in timerons) of fetching the next row for the chosen access plan up to and including this operator.
COMM_COST	DOUBLE	No	No	Estimated cumulative communication cost (in TCP/IP frames) of executing the chosen access plan up to and including this operator.
FIRST_COMM_COST	DOUBLE	No	No	Estimated cumulative communications cost (in TCP/IP frames) of fetching the first row for the chosen access plan up to and including this operator. This value includes any initial overhead required.
BUFFERS	DOUBLE	No	No	Estimated buffer requirements for this operator and its inputs.

Table 36. EXPLAIN_OPERATOR Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
REMOTE_TOTAL_COST	DOUBLE	No	No	Estimated cumulative total cost (in timerons) of performing operation(s) on remote database(s).
REMOTE_COMM_COST	DOUBLE	No	No	Estimated cumulative communication cost of executing the chosen remote access plan up to and including this operator.

Table 37. OPERATOR_TYPE Values

Value	Description
DELETE	Delete
FETCH	Fetch
FILTER	Filter rows
GENROW	Generate Row
GRPBY	Group By
HSJOIN	Hash Join
INSERT	Insert
IXAND	Dynamic Bitmap Index ANDing
IXSCAN	Index Scan
MSJOIN	Merge Scan Join
NLJOIN	Nested loop Join
RETURN	Result
RIDSCN	Row Identifier (RID) Scan
RQUERY	Remote Query
SORT	Sort
TBSCAN	Table Scan
TEMP	Temporary Table Construction
TQ	Table Queue
UNION	Union
UNIQUE	Duplicate Elimination
UPDATE	Update

EXPLAIN_PREDICATE Table

The EXPLAIN_PREDICATE table identifies which predicates are applied by a specific operator.

Table 38. EXPLAIN_PREDICATE Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.

Table 38. EXPLAIN_PREDICATE Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	INTEGER	No	No	Unique ID for this operator within this query.
PREDICATE_ID	INTEGER	No	No	Unique ID for this predicate for the specified operator.
HOW_APPLIED	CHAR(5)	No	No	How predicate is being used by the specified operator.
WHEN_EVALUATED	CHAR(3)	No	No	Indicates when the subquery used in this predicate is evaluated. Possible values are: **blank** This predicate does not contain a subquery. **EAA** The subquery used in this predicate is evaluated at application (EAA). That is, it is re-evaluated for every row processed by the specified operator, as the predicate is being applied. **EAO** The subquery used in this predicate is evaluated at open (EAO). That is, it is re-evaluated only once for the specified operator, and its results are re-used in the application of the predicate for each row. **MUL** There is more than one type of subquery in this predicate.
RELOP_TYPE	CHAR(2)	No	No	The type of relational operator used in this predicate.

Table 38. EXPLAIN_PREDICATE Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
SUBQUERY	CHAR(1)	No	No	Whether or not a data stream from a subquery is required for this predicate. There may be multiple subquery streams required. Possible values are: **N** No subquery stream is required **Y** One or more subquery streams is required
FILTER_FACTOR	DOUBLE	No	No	The estimated fraction of rows that will be qualified by this predicate.
PREDICATE_TEXT	CLOB(1M)	Yes	No	The text of the predicate as recreated from the internal representation of the SQL statement. Null if not available.

Table 39. Possible HOW_APPLIED Values

Value	Description
JOIN	Used to join tables
RESID	Evaluated as a residual predicate
SARG	Evaluated as a sargable predicate for index or data page
START	Used as a start condition
STOP	Used as a stop condition

Table 40. Possible RELOP_TYPE Values

Value	Description
blanks	Not Applicable
EQ	Equals
GE	Greater Than or Equal
GT	Greater Than
IN	In list
LE	Less Than or Equal
LK	Like
LT	Less Than
NE	Not Equal
NL	Is Null
NN	Is Not Null

EXPLAIN_STATEMENT Table

The EXPLAIN_STATEMENT table contains the text of the SQL statement as it exists for the different levels of Explain information. The original SQL statement as entered by the user is stored in this table along with the version used (by the optimizer) to choose an access plan to satisfy the SQL statement. The latter version may bear little resemblance to the original as it may have been rewritten and/or enhanced with additional predicates as determined by the SQL Compiler.

For the definition of this table, see "EXPLAIN_STATEMENT Table Definition" on page 525.

Table 41. EXPLAIN_STATEMENT Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	PK, FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	PK, FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	PK, FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	PK, FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	PK	Level of Explain information for which this row is relevant. Valid values are: **O** Original Text (as entered by user) **P** PLAN SELECTION
STMTNO	INTEGER	No	PK	Statement number within package to which this explain information is related. Set to 1 for dynamic Explain SQL statements. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS catalog view.
SECTNO	INTEGER	No	PK	Section number within package that contains this SQL statement. For dynamic Explain SQL statements, this is the section number used to hold the section for this statement at runtime. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS catalog view.
QUERYNO	INTEGER	No	No	Numeric identifier for explained SQL statement. For dynamic SQL statements (excluding the EXPLAIN SQL statement) issued through CLP or CLI, the default value is a sequentially incremented value. Otherwise, the default value is the value of STMTNO for static SQL statements and 1 for dynamic SQL statements.

Table 41. EXPLAIN_STATEMENT Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
QUERYTAG	CHAR(20)	No	No	Identifier tag for each explained SQL statement. For dynamic SQL statements issued through CLP (excluding the EXPLAIN SQL statement), the default value is 'CLP'. For dynamic SQL statements issued through CLI (excluding the EXPLAIN SQL statement), the default value is 'CLI'. Otherwise, the default value used is blanks.
STATEMENT_TYPE	CHAR(2)	No	No	Descriptive label for type of query being explained. Possible values are: **S** Select **D** Delete **DC** Delete where current of cursor **I** Insert **U** Update **UC** Update where current of cursor
UPDATABLE	CHAR(1)	No	No	Indicates if this statement is considered updatable. This is particularly relevant to SELECT statements which may be determined to be potentially updatable. Possible values are: **' '** Not applicable (blank) **N** No **Y** Yes
DELETABLE	CHAR(1)	No	No	Indicates if this statement is considered deletable. This is particularly relevant to SELECT statements which may be determined to be potentially deletable. Possible values are: **' '** Not applicable (blank) **N** No **Y** Yes
TOTAL_COST	DOUBLE	No	No	Estimated total cost (in timerons) of executing the chosen access plan for this statement; set to 0 (zero) if EXPLAIN_LEVEL is *O* (original text) since no access plan has been chosen at this time.
STATEMENT_TEXT	CLOB(1M)	No	No	Text or portion of the text of the SQL statement being explained. The text shown for the Plan Selection level of Explain has been reconstructed from the internal representation and is SQL-like in nature; that is, the reconstructed statement is not guaranteed to follow correct SQL syntax.

Table 41. EXPLAIN_STATEMENT Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
SNAPSHOT	BLOB(10M)	Yes	No	Snapshot of internal representation for this SQL statement at the Explain_Level shown. This column is intended for use with DB2 Visual Explain. Column is set to null if EXPLAIN_LEVEL is *0* (original statement) since no access plan has been chosen at the time that this specific version of the statement is captured.
QUERY_DEGREE	INTEGER	No	No	Indicates the degree of intra-partition parallelism at the time of Explain invocation. For the original statement, this contains the directed degree of intra-partition parallelism. For the PLAN SELECTION, this contains the degree of intra-partition parallelism generated for the plan to use.

EXPLAIN_STREAM Table

The EXPLAIN_STREAM table represents the input and output data streams between individual operators and data objects. The data objects themselves are represented in the EXPLAIN_OBJECT table. The operators involved in a data stream are to be found in the EXPLAIN_OPERATOR table.

Table 42. EXPLAIN_STREAM Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
STREAM_ID	INTEGER	No	No	Unique ID for this data stream within the specified operator.
SOURCE_TYPE	CHAR(1)	No	No	Indicates the source of this data stream: **O** Operator **D** Data Object

Table 42. EXPLAIN_STREAM Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
SOURCE_ID	SMALLINT	No	No	Unique ID for the operator within this query that is the source of this data stream. Set to -1 if SOURCE_TYPE is 'D'.
TARGET_TYPE	CHAR(1)	No	No	Indicates the target of this data stream: **O** Operator **D** Data Object
TARGET_ID	SMALLINT	No	No	Unique ID for the operator within this query that is the target of this data stream. Set to -1 if TARGET_TYPE is 'D'.
OBJECT_SCHEMA	VARCHAR(128)	Yes	No	Schema to which the affected data object belongs. Set to null if both SOURCE_TYPE and TARGET_TYPE are 'O'.
OBJECT_NAME	VARCHAR(128)	Yes	No	Name of the object that is the subject of data stream. Set to null if both SOURCE_TYPE and TARGET_TYPE are 'O'.
STREAM_COUNT	DOUBLE	No	No	Estimated cardinality of data stream.
COLUMN_COUNT	SMALLINT	No	No	Number of columns in data stream.
PREDICATE_ID	INTEGER	No	No	If this stream is part of a subquery for a predicate, the predicate ID will be reflected here, otherwise the column is set to -1.
COLUMN_NAMES	CLOB(1M)	Yes	No	This column contains the names and ordering information of the columns involved in this stream. These names will be in the format of: `NAME1(A)+NAME2(D)+NAME3+NAME4` Where *(A)* indicates a column in ascending order, *(D)* indicates a column in descending order, and no ordering information indicates that either the column is not ordered or ordering is not relevant.
PMID	SMALLINT	No	No	Partitioning map ID.

Table 42. EXPLAIN_STREAM Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
SINGLE_NODE	CHAR(5)	Yes	No	Indicates if this data stream is on a single or multiple partitions: **MULT** On multiple partitions; **COOR** On coordinator node; **HASH** Directed using hashing; **RID** Directed using the row ID; **FUNC** Directed using a function (PARTITION() or NODENUMBER()); **CORR** Directed using a correlation value; **Numberic** Directed to predetermined single node
PARTITION_COLUMNS	CLOB(64K)	Yes	No	List of columns this data stream is partitioned on.

ADVISE_INDEX Table

The ADVISE_INDEX table represents the recommended indexes.

Table 43. ADVISE_INDEX Table

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	No	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	No	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	No	Name of the package running when the dynamic statement was explained or name of the source file when static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	No	Schema, or qualifier, of source of Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	No	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	No	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	No	Section number within package to which this explain information is related.
QUERYNO	INTEGER	No	No	Numeric identifier for explained SQL statement. For dynamic SQL statements (excluding the EXPLAIN SQL statement) issued through CLP or CLI, the default value is a sequentially incremented value. Otherwise, the default value is the value of STMTNO for static SQL statements and 1 for dynamic SQL statements.

Table 43. ADVISE_INDEX Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
QUERYTAG	CHAR(20)	No	No	Identifier tag for each explained SQL statement. For dynamic SQL statements issued through CLP (excluding the EXPLAIN SQL statement), the default value is 'CLP'. For dynamic SQL statements issued through CLI (excluding the EXPLAIN SQL statement), the default value is 'CLI'. Otherwise, the default value used is blanks.
NAME	VARCHAR(128)	No	No	Name of the index.
CREATOR	VARCHAR(128)	No	No	Qualifier of the index name.
TBNAME	VARCHAR(128)	No	No	Name of the table or nickname on which the index is defined.
TBCREATOR	VARCHAR(128)	No	No	Qualifier of the table name.
COLNAMES	CLOB(64K)	No	No	List of column names.

UNIQUERULE	CHAR(1)	No	No	Unique rule:
				D = Duplicates allowed
				P = Primary index
				U = Unique entries only allowed

COLCOUNT	SMALLINT	No	No	Number of columns in the key plus the number of include columns if any.
IID	SMALLINT	No	No	Internal index ID.
NLEAF	INTEGER	No	No	Number of leaf pages; –1 if statistics are not gathered.
NLEVELS	SMALLINT	No	No	Number of index levels; –1 if statistics are not gathered.
FULLKEYCARD	BIGINT	No	No	Number of distinct full key values; –1 if statistics are not gathered.
FIRSTKEYCARD	BIGINT	No	No	Number of distinct first key values; –1 if statistics are not gathered.
CLUSTERRATIO	SMALLINT	No	No	Degree of data clustering with the index; –1 if statistics are not gathered or if detailed index statistics are gathered (in which case, CLUSTERFACTOR will be used instead).
CLUSTERFACTOR	DOUBLE	No	No	Finer measurement of degree of clustering, or –1 if detailed index statistics have not been gathered or if the index is defined on a nickname.
USERDEFINED	SMALLINT	No	No	Defined by the user.

Table 43. ADVISE_INDEX Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
SYSTEM_REQUIRED	SMALLINT	No	No	1 if this index is required for primary key or unique key constraint, OR if this is the index on the object identifier (OID) column of a typed table. 2 if this index is required for primary key or unique key constraint, AND this is the index on the object identifier (OID) column of a typed table. 0 otherwise.
CREATE_TIME	TIMESTAMP	No	No	Time when the index was created.
STATS_TIME	TIMESTAMP	Yes	No	Last time when any change was made to recorded statistics for this index. Null if no statistics available.
PAGE_FETCH_PAIRS	VARCHAR(254)	No	No	A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer, and the number of page fetches required to scan the table with this index using that hypothetical buffer. (Zero-length string if no data available.)
REMARKS	VARCHAR(254)	Yes	No	User-supplied comment, or null.
DEFINER	VARCHAR(128)	No	No	User who created the index.
CONVERTED	CHAR(1)	No	No	Reserved for future use.
SEQUENTIAL_PAGES	INTEGER	No	No	Number of leaf pages located on disk in index key order with few or no large gaps between them. (−1 if no statistics are available.)
DENSITY	INTEGER	No	No	Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percent (integer between 0 and 100, −1 if no statistics are available.)
FIRST2KEYCARD	BIGINT	No	No	Number of distinct keys using the first two columns of the index (−1 if no statistics or inapplicable)
FIRST3KEYCARD	BIGINT	No	No	Number of distinct keys using the first three columns of the index (−1 if no statistics or inapplicable)
FIRST4KEYCARD	BIGINT	No	No	Number of distinct keys using the first four columns of the index (−1 if no statistics or inapplicable)
PCTFREE	SMALLINT	No	No	Percentage of each index leaf page to be reserved during initial building of the index. This space is available for future inserts after the index is built.
UNIQUE_COLCOUNT	SMALLINT	No	No	The number of columns required for a unique key. Always <=COLCOUNT. < COLCOUNT only if there a include columns. −1 if index has no unique key (permits duplicates)

Table 43. ADVISE_INDEX Table (continued)

Column Name	Data Type	Nullable?	Key?	Description
MINPCTUSED	SMALLINT	No	No	If not zero, then on-line index reorganization is enabled and the value is the threshold of minimum used space before merging pages.
REVERSE_SCANS	CHAR(1)	No	No	Y = Index supports reverse scans N = Index does not support reverse scans
USE_INDEX	CHAR(1)	Yes	No	Y = index recommended or evaluated N = index not to be recommended
CREATION_TEXT	CLOB(1M)	No	No	The SQL statement used to create the index.
PACKED_DESC	BLOB(20M)	Yes	No	Internal description of the table.

ADVISE_WORKLOAD Table

The ADVISE_WORKLOAD table represents the statement that makes up the workload. For more details on workload refer to *Administration Guide: Performance*.

Table 44. ADVISE_WORKLOAD Table

Column Name	Data Type	Nullable?	Key?	Description
WORKLOAD_NAME	CHAR(128)	No	No	Name of the collection of SQL statements (workload) that this statments belongs to.
STATEMENT_NO	INTEGER	No	No	Statement number within the workload to which this explain information is related.
STATEMENT_TEXT	CLOB(1M)	No	No	Content of the SQL statement.
STATEMENT_TAG	VARCHAR(256)	No	No	Identifier tag for each explained SQL statement.
FREQUENCY	INTEGER	No	No	The number of times this statement appears within the workload.
IMPORTANCE	DOUBLE	No	No	Importance of the statement.
COST_BEFORE	DOUBLE	Yes	No	The cost (in timerons) of the query if the recommended indexes are not created.
COST_AFTER	DOUBLE	Yes	No	The cost (in timerons) of the query if the recommended indexes are created.

Table Definitions for Explain Tables

The Explain tables must be created before Explain can be invoked. The following definitions specify how to create the necessary Explain tables:
- “EXPLAIN_ARGUMENT Table Definition” on page 520
- “EXPLAIN_INSTANCE Table Definition” on page 521
- “EXPLAIN_OBJECT Table Definition” on page 522
- “EXPLAIN_OPERATOR Table Definition” on page 523
- “EXPLAIN_PREDICATE Table Definition” on page 524

- "EXPLAIN_STATEMENT Table Definition" on page 525
- "EXPLAIN_STREAM Table Definition" on page 526
- "ADVISE_INDEX Table Definition" on page 527
- "ADVISE_WORKLOAD Table Definition" on page 529

Alternately, create them by using the sample command line processor input script provided in the EXPLAIN.DDL file located in the 'misc' subdirectory of the 'sqllib' directory. Connect to the database where the Explain tables are required. Then issue the command: `db2 -tf EXPLAIN.DDL` and the tables will be created.

EXPLAIN_ARGUMENT Table Definition

```
CREATE TABLE EXPLAIN_ARGUMENT ( EXPLAIN_REQUESTER   VARCHAR(128)  NOT NULL,
                                EXPLAIN_TIME        TIMESTAMP     NOT NULL,
                                SOURCE_NAME         VARCHAR(128)  NOT NULL,
                                SOURCE_SCHEMA       VARCHAR(128)  NOT NULL,
                                EXPLAIN_LEVEL       CHAR(1)       NOT NULL,
                                STMTNO              INTEGER       NOT NULL,
                                SECTNO              INTEGER       NOT NULL,
                                OPERATOR_ID         INTEGER       NOT NULL,
                                ARGUMENT_TYPE       CHAR(8)       NOT NULL,
                                ARGUMENT_VALUE      VARCHAR(1024) NOT NULL,
                                LONG_ARGUMENT_VALUE CLOB(1M)      NOT LOGGED,
                                   FOREIGN KEY (EXPLAIN_REQUESTER,
                                                EXPLAIN_TIME,
                                                SOURCE_NAME,
                                                SOURCE_SCHEMA,
                                                EXPLAIN_LEVEL,
                                                STMTNO,
                                                SECTNO)
                                   REFERENCES EXPLAIN_STATEMENT
                                   ON DELETE CASCADE )
```

EXPLAIN_INSTANCE Table Definition

```
CREATE TABLE EXPLAIN_INSTANCE ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                SOURCE_NAME       VARCHAR(128) NOT NULL,
                                SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                EXPLAIN_OPTION    CHAR(1)      NOT NULL,
                                SNAPSHOT_TAKEN    CHAR(1)      NOT NULL,
                                DB2_VERSION       CHAR(7)      NOT NULL,
                                SQL_TYPE          CHAR(1)      NOT NULL,
                                QUERYOPT          INTEGER      NOT NULL,
                                BLOCK             CHAR(1)      NOT NULL,
                                ISOLATION         CHAR(2)      NOT NULL,
                                BUFFPAGE          INTEGER      NOT NULL,
                                AVG_APPLS         INTEGER      NOT NULL,
                                SORTHEAP          INTEGER      NOT NULL,
                                LOCKLIST          INTEGER      NOT NULL,
                                MAXLOCKS          SMALLINT     NOT NULL,
                                LOCKS_AVAIL       INTEGER      NOT NULL,
                                CPU_SPEED         DOUBLE       NOT NULL,
                                REMARKS           VARCHAR(254),
                                DBHEAP            INTEGER      NOT NULL,
                                COMM_SPEED        DOUBLE       NOT NULL,
                                PARALLELISM       CHAR(2)      NOT NULL,
                                DATAJOINER        CHAR(1)      NOT NULL,
                                    PRIMARY KEY (EXPLAIN_REQUESTER,
                                                 EXPLAIN_TIME,
                                                 SOURCE_NAME,
                                                 SOURCE_SCHEMA))
```

EXPLAIN_OBJECT Table Definition

```
CREATE TABLE EXPLAIN_OBJECT ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                              SOURCE_NAME       VARCHAR(128) NOT NULL,
                              SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                              STMTNO            INTEGER      NOT NULL,
                              SECTNO            INTEGER      NOT NULL,
                              OBJECT_SCHEMA     VARCHAR(128) NOT NULL,
                              OBJECT_NAME       VARCHAR(128) NOT NULL,
                              OBJECT_TYPE       CHAR(2)      NOT NULL,
                              CREATE_TIME       TIMESTAMP,
                              STATISTICS_TIME   TIMESTAMP,
                              COLUMN_COUNT      SMALLINT     NOT NULL,
                              ROW_COUNT         INTEGER      NOT NULL,
                              WIDTH             INTEGER      NOT NULL,
                              PAGES             INTEGER      NOT NULL,
                              DISTINCT          CHAR(1)      NOT NULL,
                              TABLESPACE_NAME   VARCHAR(128),
                              OVERHEAD          DOUBLE       NOT NULL,
                              TRANSFER_RATE     DOUBLE       NOT NULL,
                              PREFETCHSIZE      INTEGER      NOT NULL,
                              EXTENTSIZE        INTEGER      NOT NULL,
                              CLUSTER           DOUBLE       NOT NULL,
                              NLEAF             INTEGER      NOT NULL,
                              NLEVELS           INTEGER      NOT NULL,
                              FULLKEYCARD       BIGINT       NOT NULL,
                              OVERFLOW          INTEGER      NOT NULL,
                              FIRSTKEYCARD      BIGINT       NOT NULL,
                              FIRST2KEYCARD     BIGINT       NOT NULL,
                              FIRST3KEYCARD     BIGINT       NOT NULL,
                              FIRST4KEYCARD     BIGINT       NOT NULL,
                              SEQUENTIAL_PAGES  INTEGER      NOT NULL,
                              DENSITY           INTEGER      NOT NULL,
                                      FOREIGN KEY (EXPLAIN_REQUESTER,
                                                   EXPLAIN_TIME,
                                                   SOURCE_NAME,
                                                   SOURCE_SCHEMA,
                                                   EXPLAIN_LEVEL,
                                                   STMTNO,
                                                   SECTNO)
                                   REFERENCES EXPLAIN_STATEMENT
                                   ON DELETE CASCADE )
```

EXPLAIN_OPERATOR Table Definition

```
CREATE TABLE EXPLAIN_OPERATOR ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                SOURCE_NAME       VARCHAR(128) NOT NULL,
                                SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                STMTNO            INTEGER      NOT NULL,
                                SECTNO            INTEGER      NOT NULL,
                                OPERATOR_ID       INTEGER      NOT NULL,
                                OPERATOR_TYPE     CHAR(6)      NOT NULL,
                                TOTAL_COST        DOUBLE       NOT NULL,
                                IO_COST           DOUBLE       NOT NULL,
                                CPU_COST          DOUBLE       NOT NULL,
                                FIRST_ROW_COST    DOUBLE       NOT NULL,
                                RE_TOTAL_COST     DOUBLE       NOT NULL,
                                RE_IO_COST        DOUBLE       NOT NULL,
                                RE_CPU_COST       DOUBLE       NOT NULL,
                                COMM_COST         DOUBLE       NOT NULL,
                                FIRST_COMM_COST   DOUBLE       NOT NULL,
                                REMOTE_TOTAL_COST DOUBLE       NOT NULL,
                                REMOTE_COMM_COST  DOUBLE       NOT NULL,
                                     FOREIGN KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA,
                                                  EXPLAIN_LEVEL,
                                                  STMTNO,
                                                  SECTNO)
                                     REFERENCES EXPLAIN_STATEMENT
                                     ON DELETE CASCADE )
```

EXPLAIN_PREDICATE Table Definition

```
CREATE TABLE EXPLAIN_PREDICATE ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                 SOURCE_NAME       VARCHAR(128) NOT NULL,
                                 SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                 STMTNO            INTEGER      NOT NULL,
                                 SECTNO            INTEGER      NOT NULL,
                                 OPERATOR_ID       INTEGER      NOT NULL,
                                 PREDICATE_ID      INTEGER      NOT NULL,
                                 HOW_APPLIED       CHAR(5)      NOT NULL,
                                 WHEN_EVALUATED    CHAR(3)      NOT NULL,
                                 RELOP_TYPE        CHAR(2)      NOT NULL,
                                 SUBQUERY          CHAR(1)      NOT NULL,
                                 FILTER_FACTOR     DOUBLE       NOT NULL,
                                 PREDICATE_TEXT    CLOB(1M)     NOT LOGGED,
                                      FOREIGN KEY (EXPLAIN_REQUESTER,
                                                   EXPLAIN_TIME,
                                                   SOURCE_NAME,
                                                   SOURCE_SCHEMA,
                                                   EXPLAIN_LEVEL,
                                                   STMTNO,
                                                   SECTNO)
                                      REFERENCES EXPLAIN_STATEMENT
                                      ON DELETE CASCADE )
```

EXPLAIN_STATEMENT Table Definition

```
CREATE TABLE EXPLAIN_STATEMENT ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                 SOURCE_NAME       VARCHAR(128) NOT NULL,
                                 SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                 STMTNO            INTEGER      NOT NULL,
                                 SECTNO            INTEGER      NOT NULL,
                                 QUERYNO           INTEGER      NOT NULL,
                                 QUERYTAG          CHAR(20)     NOT NULL,
                                 STATEMENT_TYPE    CHAR(2)      NOT NULL,
                                 UPDATABLE         CHAR(1)      NOT NULL,
                                 DELETABLE         CHAR(1)      NOT NULL
                                 TOTAL_COST        DOUBLE       NOT NULL,
                                 STATEMENT_TEXT    CLOB(1M)     NOT NULL
                                                                NOT LOGGED,
                                 SNAPSHOT          BLOB(10M)    NOT LOGGED,
                                 QUERY_DEGREE      INTEGER      NOT NULL,
                                     PRIMARY KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA,
                                                  EXPLAIN_LEVEL,
                                                  STMTNO,
                                                  SECTNO),
                                     FOREIGN KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA)
                                     REFERENCES EXPLAIN_INSTANCE
                                     ON DELETE CASCADE )
```

EXPLAIN_STREAM Table Definition

```
CREATE TABLE EXPLAIN_STREAM ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                              SOURCE_NAME       VARCHAR(128) NOT NULL,
                              SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                              STMTNO            INTEGER      NOT NULL,
                              SECTNO            INTEGER      NOT NULL,
                              STREAM_ID         INTEGER      NOT NULL,
                              SOURCE_TYPE       CHAR(1)      NOT NULL,
                              SOURCE_ID         SMALLINT     NOT NULL,
                              TARGET_TYPE       CHAR(1)      NOT NULL,
                              TARGET_ID         SMALLINT     NOT NULL,
                              OBJECT_SCHEMA     VARCHAR(128),
                              OBJECT_NAME       VARCHAR(128),
                              STREAM_COUNT      DOUBLE       NOT NULL,
                              COLUMN_COUNT      SMALLINT     NOT NULL,
                              PREDICATE_ID      INTEGER      NOT NULL,
                              COLUMN_NAMES      CLOB(1M)     NOT LOGGED,
                              PMID              SMALLINT     NOT NULL,
                              SINGLE_NODE       CHAR(5),
                              PARTITION_COLUMNS CLOB(64K)    NOT LOGGED,
                                  FOREIGN KEY (EXPLAIN_REQUESTER,
                                               EXPLAIN_TIME,
                                               SOURCE_NAME,
                                               SOURCE_SCHEMA,
                                               EXPLAIN_LEVEL,
                                               STMTNO,
                                               SECTNO)
                                  REFERENCES EXPLAIN_STATEMENT
                                  ON DELETE CASCADE )
```

ADVISE_INDEX Table Definition

```
CREATE TABLE ADVISE_INDEX (EXPLAIN_REQUESTER VARCHAR(128) NOT NULL
                                                WITH DEFAULT '',
                           EXPLAIN_TIME      TIMESTAMP    NOT NULL
                                                WITH DEFAULT CURRENT TIMESTAMP,
                           SOURCE_NAME       VARCHAR(128) NOT NULL
                                                WITH DEFAULT '',
                           SOURCE_SCHEMA     VARCHAR(128) NOT NULL
                                                WITH DEFAULT '',
                           EXPLAIN_LEVEL     CHAR(1)      NOT NULL
                                                WITH DEFAULT '',
                           STMTNO            INTEGER      NOT NULL
                                                WITH DEFAULT 0,
                           SECTNO            INTEGER      NOT NULL
                                                WITH DEFAULT 0,
                           QUERYNO           INTEGER      NOT NULL
                                                WITH DEFAULT 0,
                           QUERYTAG          CHAR(20)     NOT NULL
                                                WITH DEFAULT '',
                           NAME              VARCHAR(128) NOT NULL,
                           CREATOR           VARCHAR(128) NOT NULL
                                                WITH DEFAULT '',
                           TBNAME            VARCHAR(128) NOT NULL,
                           TBCREATOR         VARCHAR(128) NOT NULL
                                                WITH DEFAULT '',
                           COLNAMES          CLOB(64K)    NOT NULL,
                           UNIQUERULE        CHAR(1)      NOT NULL
                                                WITH DEFAULT '',
                           COLCOUNT          SMALLINT     NOT NULL
                                                WITH DEFAULT 0,
                           IID               SMALLINT     NOT NULL
                                                WITH DEFAULT 0,
                           NLEAF             INTEGER      NOT NULL
                                                WITH DEFAULT 0,
                           NLEVELS           SMALLINT     NOT NULL
                                                WITH DEFAULT 0,
                           FIRSTKEYCARD      BIGINT       NOT NULL
                                                WITH DEFAULT 0,
                           FULLKEYCARD       BIGINT       NOT NULL
                                                WITH DEFAULT 0,
                           CLUSTERRATIO      SMALLINT     NOT NULL
                                                WITH DEFAULT 0,
                           CLUSTERFACTOR     DOUBLE       NOT NULL
                                                WITH DEFAULT 0,
                           USERDEFINED       SMALLINT     NOT NULL
                                                WITH DEFAULT 0,
                           SYSTEM_REQUIRED   SMALLINT     NOT NULL
                                                WITH DEFAULT 0,
                           CREATE_TIME       TIMESTAMP    NOT NULL
                                                WITH DEFAULT CURRENT TIMESTAMP,
                           STATS_TIME        TIMESTAMP
                                                WITH DEFAULT CURRENT TIMESTAMP,
                           PAGE_FETCH_PAIRS  VARCHAR(254) NOT NULL
                                                WITH DEFAULT '',
                           REMARKS           VARCHAR(254)
                           WITH DEFAULT '',
DEFINER            VARCHAR(128) NOT NULL
                           WITH DEFAULT '',
CONVERTED           CHAR(1) NOT NULL
                           WITH DEFAULT '',
SEQUENTIAL_PAGES   INTEGER      NOT NULL
                           WITH DEFAULT 0,
DENSITY            INTEGER      NOT NULL
                           WITH DEFAULT 0,
FIRST2KEYCARD      BIGINT       NOT NULL
                           WITH DEFAULT 0,
FIRST3KEYCARD      BIGINT       NOT NULL
                           WITH DEFAULT 0,
FIRST4KEYCARD      BIGINT       NOT NULL
                           WITH DEFAULT 0,
PCTFREE            SMALLINT     NOT NULL
                           WITH DEFAULT -1,
UNIQUE_COLCOUNT    SMALLINT     NOT NULL
                           WITH DEFAULT -1,
MINPCTUSED         SMALLINT     NOT NULL
                           WITH DEFAULT 0,
REVERSE_SCANS      CHAR(1)      NOT NULL
                           WITH DEFAULT 'N',
USE_INDEX          CHAR(1),
CREATION_TEXT      CLOB(1M)     NOT NULL
                           NOT LOGGED WITH DEFAULT '',
PACKED_DESC        BLOB(1M)     NOT LOGGED)
```

ADVISE_WORKLOAD Table Definition

```
CREATE TABLE ADVISE_WORKLOAD (WORKLOAD_NAME  CHAR(128)    NOT NULL
                                                WITH DEFAULT 'WK0',
                              STATEMENT_NO   INTEGER      NOT NULL
                                                WITH DEFAULT 1,
                              STATEMENT_TEXT CLOB(1M)     NOT NULL NOT LOGGED,
                              STATEMENT_TAG  VARCHAR(256) NOT NULL
                                                WITH DEFAULT '',
                              FREQUENCY      INTEGER      NOT NULL
                                                WITH DEFAULT 1,
                              IMPORTANCE     DOUBLE       NOT NULL
                                                WITH DEFAULT 1,
                              COST_BEFORE    DOUBLE,
                              COST_AFTER     DOUBLE)
```

Appendix C. SQL Explain Tools

The **db2expln** tool describes the access plan selected for static SQL statements in the packages stored in the system catalog tables. It can be used to obtain a quick explanation of the chosen access plan for packages for which explain data was not captured at bind time.

The **dynexpln** tool describes the access plan selected for dynamic statements. It creates a static package for the statements and then uses the **db2expln** tool to describe them.

You can use these Explain tools to understand the access plan chosen for a particular SQL statement. Or, you could use the integrated Explain Facility ("Chapter 7. SQL Explain Facility" on page 201) in conjunction with Visual Explain to understand the access plan chosen for a particular SQL statement. Both dynamic and static SQL statements can be explained using the Explain Facility. One difference from the Explain tools is that with Visual Explain the Explain information is presented in a graphical format. Otherwise the level of detail provided in the two methods is equivalent.

To fully use the output of `db2expln`, and `dynexpln` you must understand:

- The different SQL statements supported and the terminology related to those statements (such as predicates in a SELECT statement).
- The purpose of a package (access plan). (See "Data Access Concepts and Optimization" on page 153 for this information.)
- The purpose and contents of the system catalog tables. (Refer to the *SQL Reference* for this information.)
- Other concepts described in "Part 2. Tuning Application Performance" on page 39.

The following topics provide information about `db2expln` and `dynexpln`:

- Running db2expln and dynexpln
- db2expln Syntax and Parameters
- Usage Notes for db2expln
- dynexpln Syntax and Parameters
- Usage Notes for dynexpln
- Description of db2expln and dynexpln Output
- Examples of db2expln and dynexpln Output.

Running db2expln and dynexpln

The explain tools (`db2expln` and `dynexpln`) are located in the `misc` subdirectory of your instance `sqllib` directory. If `db2expln` and `dynexpln` are not in your current directory, they must be in a directory that appears in your PATH environment variable.

The `db2expln` program connects and binds itself to a database using the `db2expln.bnd` file the first time the database is accessed. The `db2expln.bnd` file is in the `bnd` subdirectory of your `sqllib` directory.

To run `db2expln`, you must have SELECT privilege to the system catalog views as well as EXECUTE authority for the `db2expln` package. To run `dynexpln`, you must have BINDADD authority for the database as well as any privileges needed for the SQL statements being explained. (Note that if you have SYSADM or DBADM authority, you will automatically have all these authorization levels.)

db2expln Syntax and Parameters

```
►►──db2expln──┬──────────────┬──┬─────────────────────┬──┬──────────────────────────┬──┬────┬──┬────┬──►
              └─-c──creator──┘  └─-d──database name──┘  └─-e──escape character──┘  └─-g─┘  ├─-h─┤
                                                                                            └─-?─┘

►──┬────┬──┬────┬──┬─────────────────┬──┬────────────────────┬──┬──────────────────────┬──────────────►
   └─-i─┘  └─-l─┘  ├─-o──output file─┤  └─-p──package name──┘  └─-s──section number──┘
                   └─-t──────────────┘

►──┬──────────────────────────┬───────────────────────────────────────────────────────────────────►◄
   └─-u──userID──password──┘
```

Where:

-c creator
: The user ID of the package creator.

  If you do not specify this option, you will be prompted for it.

  You may specify the creator name using the pattern matching characters, percent sign (%) and underscore (_) that may be used in a LIKE predicate.

-d database name
: The name of the database that contains the packages to be explained.

  If you do not specify this option, you will be prompted for it.

-e escape character
: Used to specify the character that is to be interpreted as an escape character, rather than a pattern-matching character.

  For example, the `db2expln` command to explain the package TESTID.CALC% is `db2expln -c TESTID -p CALC%.` However, this command would also explain any other plans that start with `CALC`. To explain just the TESTID.CALC% package, you must use an escape character. By changing the command to read: `db2expln -c TESTID -e ! -p CALC!%` you specify that the `!` character will be used as an escape character and `!%` is interpreted as the `%` character.

-g
: Show optimizer plan graphs. Each section is examined, and the original optimizer plan graph (as presented by Visual Explain) is constructed. Note that the generated graph may not match the original plan.

-h or -?
: Obtain help information about the input parameters. Specifying this option overrides all other options.

-i
: Display operator IDs in the explained plan. The operator IDs allow the output from `db2expln` to be matched to the output from the Explain facility.

-l
: The package name can be either lower or mixed-case if this option is specified. If this **-l** option is not specified, the package name is converted to uppercase

-o output file
: The name of the file to which `db2expln` will write the results.

  If you specify **-o** without a file name, you will be prompted for a file name. The default file name is `db2expln.out`.

-p package name
: The name of the package to be explained.

  If you do not specify this option you will be prompted to provide it.

  You may specify the package name using the pattern matching characters, percent sign (%) and underscore (_) that can be used in a LIKE predicate.

-s section number
: The section number to explain within the package. The number zero (0) may be specified if you wish to have all sections in the package explained. If the package creator (**-c**) or package name (**-p**) arguments imply that multiple packages will be explained, and thus multiple sections, the section value, if provided, is overridden with a zero (0).

  If you do not specify this option you will be prompted to provide it.

Section numbers can be found by querying the system catalog SYSCAT.STATEMENTS (Refer to the *SQL Reference* for a description of the system catalog tables.)

-t The output is directed to the terminal.

If you do not specify **-o** or **-t**, you will be prompted for a file name, with the default displaying the output at the terminal.

-u userID password
When connecting to a database, use the provided user ID and password.

Both the user id and password must be valid according to naming conventions and be recognized by the database.

Some of the option flags above may have special meaning to your operating system and, as a result, may not be interpreted correctly in the `db2expln` command line. However, it may be possible to enter these characters by preceding them with an escape character. For more information, see your operating system user's manual.

Help and initial status messages, produced by `db2expln`, are written to standard output. All prompts and other status messages produced by the explain tool are written to standard error. Explain text is written to standard output or to a file depending on the output option chosen.

With the **-p** and **-c** options, multiple plans can be explained with one invocation of explain by specifying string constants for packages and creators with LIKE patterns. That is, the underscore (_) may be used to represent a single character, and the percent sign (%) may be used to represent the occurrence of zero or more characters.

For example, to explain all sections for all packages in a database named SAMPLE, with the results being written to the file **my.exp**, enter

```
db2expln -d SAMPLE -p % -c % -s 0 -o my.exp
```

Usage Notes for db2expln

The following are common messages displayed by `db2expln`:

- `No packages found for database <database>, package pattern: <creator>.<package>.`

  This message will appear in the output if no packages were found in the database that matched the specified pattern.
- `Bind messages can be found in db2expln.msg`

This message will appear in the output if the bind of `db2expln.bnd` was not successful. Further information on the problems encountered will be found in the file `db2expln.msg` in the current directory.

- `Section number overridden to 0 for potential multiple packages.`

  This message will appear in the output if multiple packages may be encountered by `db2expln`. This action will be taken if one of the pattern matching characters is used in the package or creator input arguments.
- `No static sections qualify from package.`

  This message will appear in the output if the specified package only contains dynamic SQL statements which means that there are no static sections.
- `Database <database>, package <creator>.<package> is not valid. Rebind and then rerun db2expln.`

  This message will appear in the output if the package specified is currently not valid. As directed, reissue the BIND or REBIND command for the plan to re-create a valid package in the database, and then rerun `db2expln`.
- `Section not processed: Produced by unsupported release.`

  This message will also appear in the output if the section currently being processed was produced by a release of DB2 other than the one for which this `db2expln` executable was provided. In this case, use the copy of `db2expln` from the release of DB2 that produced the section.

SQL Statements Excluded: The following statements will not be explained:
- BEGIN/END DECLARE SECTION
- BEGIN/END COMPOUND
- INCLUDE
- WHENEVER
- COMMIT and ROLLBACK
- CONNECT
- OPEN cursor
- FETCH
- CLOSE cursor
- PREPARE
- EXECUTE
- EXECUTE IMMEDIATE
- DESCRIBE
- Dynamic DECLARE CURSOR
- SQL control statements

Each sub-statement within a compound SQL statement may have its own section, which can be explained by **db2expln**.

dynexpln Syntax and Parameters

Where:

```
►►──dynexpln──┬──────────────────────┬──┬──────────────────┬──┬──────┬──┬──────┬──┬──────┬─────────────────────►
              └─-d──database name────┘  └─-f──input file───┘  └─-g───┘  ├─-h───┤  └─-i───┘
                                                                         └─-?───┘

►──┬──────────────────┬──┬──────────────────────┬──┬────────────────────────┬──────────────────────────────────►
   ├─-o──output file──┤  └─-q──SQL statement────┘  └─-u──userID password────┘
   └─-t───────────────┘

►──┬─────────────────────────────┬────────────────────────────────────────────────────────────────────────────►◄
   └─-z──statement terminator────┘
```

-d database name
: The name of the database that contains the packages to be explained.

  If you do not specify this option, you will be prompted for it.

-f input file
: The name of the file which contains the SQL statements to be explained.

  Unless you use the statement terminator (**-e**) option, only one SQL statement should appear on each line of the file. SQL comments may be entered into the file. An SQL comment starts with `--` and goes to the end of the line.

-g
: Show optimizer plan graphs. Each section is examined, and the original optimizer plan graph (as presented by Visual Explain) is constructed. Note that the generated graph may not match the original plan.

-h or -?
: Obtain help information about the input parameters. Specifying this option overrides all other options.

-i
: Display operator IDs in the explained plan. The operator IDs allow the output from `db2expln` to be matched to the output from the Explain facility.

-o output file
: The name of the file to which `db2expln` will write the results.

-q SQL statement
: The SQL statement to be explained.

  If you do not specify this option and you do not specify the input file (**-f**) optional parameter, you will be prompted to provide the SQL statement to be explained.

If you specify both this option and the input file (**-f**) optional parameter, `dynexpln` will first describe the statements provided by the SQL statement (**-s**) option and then describe the statements in the input file (**-f**).

-t The output is directed to the terminal.

If both the output (**-o**) and **-t** options are specified, then the output is directed to the terminal.

If you do not specify the output file (**-o**) or **-t** options, you will be prompted for a file name, with the default displaying the output at the terminal.

-u userID password
When connecting to a database, use the provided user ID and password.

Both the user id and password must be valid according to naming conventions and be recognized by the database.

-z statement terminator
The character used to indicate that the end of an SQL statement has been reached.

The default is that there is no statement terminator. By not using this option, each line of the file will be assumed to be a separate SQL statement. If you use this option, `dynexpln` will use the specified termination character to separate the statements.

Some of the option flags above may have special meaning to your operating system and, as a result, may not be interpreted correctly in the `dynexpln` command line. However, it may be possible to enter these characters by preceding them with an escape character. For more information, see your operating system user's manual.

If you use the statement terminator (**-e**) option, you may enter multiple statements using the SQL statement (**-s**) option. If you do this, you should separate the statements with the termination character.

Help and initial status messages, produced by `dynexpln`, are written to standard output. All prompts and other status messages produced by the explain tool are written to standard error. Explain text is written to standard output or to a file depending on the output option chosen.

For example, to connect to a database named SAMPLE and explain all the statements in the file `TRYIT`, with the results being written to the file `my.exp`, enter

```
dynexpln -d SAMPLE -f TRYIT -o my.exp
```

Usage Notes for dynexpln

To explain dynamic statements, `dynexpln` creates a static application for the statements and then invokes `db2expln`. To create the static statements, `dynexpln` generates a trivial C program with the statements and then calls the DB2 precompiler to create the package. (The generated C program is not complete and cannot be compiled; it only contains enough information that the precompiler can build the package.)

The following are common messages displayed by `dynexpln`:

- All error messages from `db2expln`.

  Since `dynexpln` invokes `db2expln`, it is possible to see most of `db2expln`'s error messages.
- `Error connecting to the database.`

  This message will appear in the output if an error occurred connecting to the database. A CLI error message will also be displayed indicating why the connection could not be completed. Correct the cause of the error and run `dynexpln` again.
- `The file "<filename>" must be removed before dynexpln will run.`

  This message will appear if the given file exists at the time `dynexpln` is run. Remove the file or change the value of the `DYNEXPLN_PACKAGE` environment variable to change the name of the file which will be created and run `dynexpln` again.
- `The package "<creator>.<package>" must be dropped before dynexpln will run.`

  This message will appear if the given package exists at the time `dynexpln` is run. Drop the package and run or change the value of the **DYNEXPLN_PACKAGE** environment variable to change the name of the package which will be created and run `dynexpln` again.
- `Error writing file "<filename>".`

  This message will appear if the given file cannot be written to. Ensure that `dynexpln` can write files in the current directory and run it again.
- `Error reading input file "<filename>".`

  This message will appear if the file given with the **-f** option cannot be read from. Ensure that the file exists and that `dynexpln` can read it. Then run `dynexpln` again.

Environment Variables: There are two different environment variables that can be used in conjunction with `dynexpln`:

- **DYNEXPLN_OPTIONS** are the SQL precompiler options you use when building the package for your statements. Use the same syntax variable as you would when issuing a PREP command through CLP.

  For example: `DYNEXPLN_OPTIONS="OPTLEVEL 5 BLOCKING ALL"`

- **DYNEXPLN_PACKAGE** is the name of the package which is created in the database. The statements to be described are placed in this package. If this variable is not defined, the package is given a default value of **DYNEXPLN**. (Only the first eight characters of the name in this environment variable are used.)

  The name is also used to create the names for the intermediate files that `dynexpln` uses.

Description of db2expln and dynexpln Output

In the output, the explain information for each package is broken into two parts:

- Package information such as date of bind and relevant bind options
- Section information such as the section number followed by the SQL statement being explained. Beneath the section information will be the explain output of the access plan chosen for the SQL statement shown.

The steps of an access plan, or section, will be presented in the order that the database manager executes them. Each major step will be shown as a left-justified heading with information about that step indented beneath it. The explain output for the access plan has indentation bars provided in the left margin of the output. These bars also provide the ″scope″ for the operation; operations at a lower (that is, further to the right) level of indentation within the same operation are processed before returning to the previous level of indentation.

It is important to remember that the access plan chosen was based on an augmented version of the original SQL statement (the one shown in the output). For example, the original statement may cause any number of triggers and constraints to be activated. As well, the SQL statement may be rewritten to an equivalent but more efficient format by the Query Rewrite component of the SQL Compiler. All of these factors are included in the information presented to the Optimizer when it determines the most efficient plan to satisfy the statement. Thus, the access plan shown in the explain output may differ substantially from the access plan that one might expect for the original SQL statement. The integrated Explain facility (see “Chapter 7. SQL Explain Facility” on page 201) shows the actual SQL statement used for optimization in the form of an SQL-like statement which is created by reverse-translating the internal representation of the query.

When comparing output from `db2expln` or `dynexpln` to the output of the Explain facility, the operator ID option (**-i**) can be very useful. Each time `db2expln` or `dynexpln` starts processing a new operator from the Explain facility, the operator ID number will be printed to the left of the explained plan. The operator IDs can be used to match up the steps in the different representations of the access plan. Note that there is not always a one-to-one correspondence between the operators in the Explain facility output and the operations shown by `db2expln` and `dynexpln`.

The following topics describe the explain text that may be produced by `db2expln` and `dynexpln`:
- Table Access
- Temporary Tables
- Joins
- Data Streams
- Insert, Update, and Delete
- Row Identifier (RID) Preparation
- Aggregation
- Parallel Processing
- Federated Statement Processing
- Miscellaneous Statements.

Table Access

This statement tells the name and type of table being accessed. It has two formats that are used:

1. Regular tables of three types:
   - Access Table Name:
     ```
     Access Table Name = schema.name  ID = ts,n
     ```
     where:
     – *schema.name* is the fully-qualified name of the table being accessed
     – *ID* is the corresponding TABLESPACEID and TABLEID from the SYSCAT.TABLES catalog for the table
   - Access Hierarchy Table Name:
     ```
     Access Hierarchy Table Name = schema.name  ID = ts,n
     ```
     where:
     – *schema.name* is the fully-qualified name of the table being accessed
     – *ID* is the corresponding TABLESPACEID and TABLEID from the SYSCAT.TABLES catalog for the table
   - Access Summary Table Name:
     ```
     Access Summary Table Name = schema.name  ID = ts,n
     ```
     where:
     – *schema.name* is the fully-qualified name of the table being accessed

   - *ID* is the corresponding TABLESPACEID and TABLEID from the SYSCAT.TABLES catalog for the table
2. Temporary tables of two types:
   - Access Temporary Table ID:
     ```
     Access Temp Table  ID = tn
     ```
     where:
     - *ID* is the corresponding identifier assigned by `db2expln`
   - Access Declared Global Temporary Table ID:
     ```
     Access Global Temp Table  ID = ts,tn
     ```
     where:
     - *ID* is the corresponding TABLESPACEID from the SYSCAT.TABLES catalog for the table (`ts`); and the corresponding identifier assigned by `db2expln` (`tn`)

Following the table access statement, additional statements will be provided to further describe the access. These statements will be indented under the table access statement. The possible statements are:
- Number of Columns
- Parallel Scan
- Scan Direction
- Row Access Method
- Lock Intents
- Predicates
- Miscellaneous Table Statements.

Number of Columns
The following statement indicates the number of columns being used from each row of the table:
```
#Columns = n
```

Parallel Scan
The following statement indicates that the database manager will use several subagents to read from the table in parallel:
```
Parallel Scan
```

If this text is not shown, the table will only be read from by one agent (or subagent).

Scan Direction
The following statement indicates that the database manager will read rows in a reverse order:

```
Scan Direction = Reverse
```

If this text is not shown, the scan direction is forward, which is the default.

Row Access Method

One of the following statements will be displayed, indicating how the qualifying rows in the table are being accessed:

- The `Relation Scan` statement indicates that the table is being sequentially scanned to find the qualifying rows.
  - The following statement indicates that no prefetching of data will be done:

    ```
    Relation Scan
    |  Prefetch: None
    ```
  - The following statement indicates that the optimizer has predetermined the number of pages that will be prefetched:

    ```
    Relation Scan
    |  Prefetch: n Pages
    ```
  - The following statement indicates that data should be prefetched:

    ```
    Relation Scan
    |  Prefetch: Eligible
    ```
  - The following statement indicates that the qualifying rows are being identified and accessed through an index:

    ```
    Index Scan:  Name = schema.name  ID = xx
    |  Index Columns:
    ```

    where:
    - *schema.name* is the fully-qualified name of the index being scanned
    - *ID* is the corresponding IID column in the SYSCAT.INDEXES catalog view.

    This will be followed by one row for each column in the index. Each column in the index will be listed in one of the following forms:

    ```
    n: column_name (Ascending)
    n: column_name (Descending)
    n: column_name (Include Column)
    ```

    The following statements are provided to clarify the type of index scan:
    - The range delimiting predicates for the index are shown by:

      ```
      #Key Columns = n
      |  Start Key: xxxxx
      |  Stop Key: xxxxx
      ```

      Where `xxxxx` is one of:

        - `Start of Index`
        - `End of Index`
        - `Inclusive Value:` or `Exclusive  Value:`

          An inclusive key value will be included in the index scan. An exclusive key value will not be included in the scan. The value for the key will be given by one of the following rows for each part of the key:

          ```
          n: 'string'
          n: nnn
          n: yyyy-mm-dd
          n: hh:mm:ss
          n: yyyy-mm-dd hh:mm:ss.uuuuuu
          n: NULL
          n: ?
          ```

          If a literal string is shown, on the first 20 characters are displayed. If the string is longer than 20 characters, this will be shown by `...` at the end of the string. Some keys cannot be determined until the section is executed. This is shown by a `?` as the value.
    - `Index-Only Access`

      If all the needed columns can be obtained from the index key, this statement will appear and no table data will be accessed.
    - The following statement indicates that no prefetching of index pages will be done:

      ```
      Index Prefetch: None
      ```
    - The following statement indicates that index pages should be prefetched:

      ```
      Index Prefetch: Eligible
      ```
    - The following statement indicates that no prefetching of data pages will be done:

      ```
      Data Prefetch: None
      ```
    - The following statement indicates that data pages should be prefetched:

      ```
      Data Prefetch: Eligible
      ```
    - If there are predicates that can be passed to the Index Manager to help qualify index entries, the following statement is used to show the number of predicates:

      ```
      Sargable Index Predicate(s)
      |  #Predicates = n
      ```
- The `Fetch Direct` statement indicates that the qualifying rows are being accessed by using row IDs (RIDs) that were prepared earlier in the access plan.

Lock Intents

For each table access, the type of lock that will be acquired at the table and row levels is shown with the following statement:

```
Lock Intents
|  Table: xxxx
|  Row  : xxxx
```

Possible values for a table lock are:
- Exclusive
- Intent Exclusive
- Intent None
- Intent Share
- Share
- Share Intent Exclusive
- Super Exclusive
- Update

Possible values for a row lock are:
- Exclusive
- Next Key Exclusive (does not appear in `db2expln` output)
- None
- Share
- Next Key Share
- Update
- Next Key Weak Exclusive
- Weak Exclusive

The explanation of these lock types is found in "Attributes of Locks" on page 49.

Predicates

There are two statements that provide information about the predicates used in an access plan:

1. The following statement indicates the number of predicates that will be evaluated once the data has been returned:
   ```
   Residual Predicate(s)
   |  #Predicates = n
   ```
2. The following statement indicates the number of predicates that will be evaluated while the data is being accessed. The count of predicates does not include push-down operations such as aggregation or sort.
   ```
   Sargable Predicate(s)
   |  #Predicates = n
   ```

The number of predicates shown in the above statements may not reflect the number of predicates provided in the SQL statement because predicates can be:

- Applied more than once within the same query
- Transformed and extended with the addition of implicit predicates during the query optimization process
- Transformed and condensed into fewer predicates during the query optimization process.

Miscellaneous Table Statements

- The following statement indicates that only one row will be accessed:

  ```
  Single Record
  ```

- The following statement appears when the isolation level used for this table access uses a different isolation level than the package:

  ```
  Isolation Level: xxxx
  ```

  A different isolation level may be used for a number of reasons, including:

  – A package was bound with Repeatable Read and affects referential integrity constraints; the access of the parent table to check referential integrity constraints is downgraded to an isolation level of Cursor Stability to avoid holding unnecessary locks on this table.

  – A package bound with Uncommitted Read issues a DELETE or UPDATE statement; the table access for the actual delete is upgraded to Cursor Stability.

- The following statement indicates that some or all of the rows read from the temporary table will be cached outside the buffer pool if sufficient sortheap memory is available:

  ```
  Keep Rows In Private Memory
  ```

- If the table has the volatile cardinality attribute set, it will be indicated by:

  ```
  Volatile Cardinality
  ```

Temporary Tables

A temporary table is used by an access plan to store data during its execution in a transient or temporary work table. This table only exists while the access plan is being executed. Generally, temporary tables are used when subqueries need to be evaluated early in the access plan, or when intermediate results will not fit in the available memory.

If a temporary table needs to be created, then one of two possible statements may appear. These statements indicate that a temporary table is to be created and rows inserted into it. The ID is an identifier assigned by `db2expln` for convenience when referring to the temporary table. This ID is prefixed with the letter ’t’ to indicate that the table is a temporary table.

- The following statement indicates an ordinary temporary table will be created:
  ```
  Insert Into Temp Table  ID = tn
  ```
- The following statement indicates an ordinary temporary table will be created by multiple subagents in parallel:
  ```
  Insert Into Shared Temp Table  ID = tn
  ```
- The following statement indicates a sorted temporary table will be created:
  ```
  Insert Into Sorted Temp Table  ID = tn
  ```
- The following statement indicates a sorted temporary table will be created by multiple subagents in parallel:
  ```
  Insert Into Sorted Shared Temp Table  ID = tn
  ```
- The following statement indicates a declared global temporary table will be created:
  ```
  Insert Into Global Temp Table  ID = ts,tn
  ```
- The following statement indicates a declared global temporary table will be created by multiple subagents in parallel:
  ```
  Insert Into Shared Global Temp Table  ID = ts,tn
  ```
- The following statement indicates a sorted declared global temporary table will be created:
  ```
  Insert Into Sorted Global Temp Table  ID = ts,tn
  ```
- The following statement indicates a sorted declared global temporary table will be created by multiple subagents in parallel:
  ```
  Insert Into Sorted Shared Global Temp Table  ID = ts,tn
  ```

Each of the above statements will be followed by:

```
#Columns = n
```

which indicates how many columns are in each row being inserted into the temporary table.

Sorted Temporary Tables

Sorted temporary tables can result from such operations as:

- ORDER BY
- DISTINCT
- GROUP BY
- Merge Join
- '= ANY' subquery
- '<> ALL' subquery
- INTERSECT or EXCEPT
- UNION (without the ALL keyword)

A number of additional statements may follow the original creation statement for a sorted temporary table:

- The following statement indicates the number of key columns used in the sort:

  ```
  #Sort Key Columns = n
  ```

  For each column in the sort key, one of the following lines will be displayed:

  ```
  Key n: column_name (Ascending)
  Key n: column_name (Descending)
  Key n: (Ascending)
  Key n: (Descending)
  ```
- The following statements provide estimates of the number of rows and the row size so that the optimal sort heap can be allocated at run time.

  ```
  Sortheap Allocation Parameters:
  |  #Rows     = n
  |  Row Width = n
  ```
- If only the first rows of the sorted result are needed, the following is displayed:

  ```
  Sort Limited To Estimated Row Count
  ```
- For sorts in a symmetric multiprocessor (SMP) environment, the type of sort to be performed is indicated by one of the following statements:

  ```
  Use Partitioned Sort
  Use Shared Sort
  Use Replicated Sort
  Use Round-Robin Sort
  ```

  For a description of the different sorting techniques, see “Parallel Sort Strategies” on page 183.
- The following statements indicate whether or not the result from the sort will be left in the sort heap:

  ```
  Piped
  ```

  and

  ```
  Not Piped
  ```

  If a piped sort is indicated, the database manager will keep the sorted output in memory, rather than placing the sorted result in another temporary table. (For a description of piped versus non-piped sorts, see “Influence of Sorting on the Optimizer” on page 180.)
- The following statement indicates that duplicate values will be removed during the sort:

  ```
  Duplicate Elimination
  ```

- If aggregation is being performed in the sort, it will be indicated by one of the following statements:

```
Partial Aggregation
Intermediate Aggregation
Buffered Partial Aggregation
Buffered Intermediate Aggregation
```

Temporary Table Completion

After a table access that contains a push-down operation to create a temporary table (that is, a create temporary table that occurs within the scope of a table access), there will be a ″completion″ statement, which handles end-of-file by getting the temporary table ready to provide rows to subsequent temporary table access. One of the following lines will be displayed:

```
Temp Table Completion  ID = tn
Shared Temp Table Completion  ID = tn
Sorted Temp Table Completion  ID = tn
Sorted Shared Temp Table Completion  ID = tn
```

Table Functions

Table functions are user defined functions (UDFs) that return data to the statement in the form of a table. Refer to the *SQL Reference* for more information about table functions. Table functions are indicated by the statement:

```
Access User Defined Table Function
|   Name = schema.funcname
|   Language = xxxx
|   Fenced   Deterministic   NULL Call   Disallow Parallel
```

The language (C, OLE, or Java) that the table function is written in is given along with the attributes of the table function.

Joins

There are three types of joins (see “Join Concepts” on page 165 for a description of these joins):

- Hash join
- Merge join
- Nested loop join.

When the time comes in the execution of a section for a join to be performed, one of the following statements is displayed:

```
Hash Join
```

or

```
Merge Join
```

or

```
   Nested Loop Join
```

It is possible for a left outer join to be performed. A left outer join is indicated by one of the following statements:

```
   Left Outer Hash Join
```

or

```
   Left Outer Merge Join
```

or

```
   Left Outer Nested Loop Join
```

For merge and nested loop joins, the outer table of the join will be the table referenced in the previous access statement shown in the output. The inner table of the join will be the table referenced in the access statement that is contained within the scope of the join statement. For hash joins, the access statements are reversed with the outer table contained within the scope of the join and the inner table appearing before the join.

For a hash or merge join, the following additional statements may appear:

- In some circumstances, a join simply needs to determine if any row in the inner table matches the current row in the outer. This is indicated with the statement:
  ```
  Early Out: Single Match Per Outer Row
  ```
- It is possible to apply predicates after the join has completed. The number of predicates being applied will be indicated as follows:
  ```
  Residual Predicate(s)
  |  #Predicates = n
  ```

For a hash join, the following additional statements may appear:

- The hash table is built from the inner table. If the hash table build was pushed down into a predicate on the inner table access, it is indicated by the following statement in the access of the inner table:
  ```
     Process Hash Table For Join
  ```
- While accessing the outer table, a probe table can be built to improve the perfromance of the join. The probe table build is indicated by the following statement in the access of the outer table:
  ```
     Process Probe Table For Hash Join
  ```
- The estimated number of bytes needed to build the hash table is represented by:
  ```
     Estimated Build Size: n
  ```

- The estimated number of bytes needed for the probe table is represented by:

  ```
  Estimated Probe Size: n
  ```

For a nested loop join, the following additional statement may appear immediately after the join statement:

```
Piped Inner
```

This statement indicates that the inner table of the join is the result of another series of operations. This is also referred to as a *composite inner*.

If a join involves more than two tables, the explain steps should be read from top to bottom. For example, suppose the explain output has the following flow:

```
Access ..... W
Join
|  Access ..... X
Join
|  Access ..... Y
Join
|  Access ..... Z
```

The steps of execution would be:

1. Take a row that qualifies from `W`.
2. Join row from `W` with (next) row from `X` and call the result `P1` (for partial join result number 1).
3. Join `P1` with (next) row from `Y` to create `P2`.
4. Join `P2` with (next) row from `Z` to obtain one complete result row.
5. If there are more rows in `Z`, go to step 4.
6. If there are more rows in `Y`, go to step 3.
7. If there are more rows in `X`, go to step 2.
8. If there are more rows in `W`, go to step 1.

Data Streams

Within an access plan, there is often a need to control the creation and flow of data from one series of operations to another. The data stream concept allows a group of operations within an access plan to be controlled as a unit. The start of a data stream is indicated by the following statement:

```
Data Stream n
```

where `n` is a unique identifier assigned by `db2expln` for ease of reference. The end of a data stream is indicated by:

```
End of Data Stream n
```

All operations between these statements are considered part of the same data stream.

A data stream has a number of characteristics and one or more statements can follow the initial data stream statement to describe these characteristics:

- If the operation of the data stream depends on a value generated earlier in the access plan, the data stream is marked with:
  ```
  Correlated
  ```
- Similar to a sorted temporary table, the following statements indicate whether or not the results of the data stream will be kept in memory:
  ```
  Piped
  ```
  and
  ```
  Not Piped
  ```
  As was the case with temporary tables, a piped data stream may be written to disk, if insufficient memory exists at execution time. The access plan will provide for both possibilities.
- The following statement indicates that only a single record is required from this data stream:
  ```
  Single Record
  ```

When a data stream is accessed, the following statement will appear in the output:

```
Access Data Stream n
```

Insert, Update, and Delete

The explain text for these SQL statements is self-explanatory. Possible statement text for these SQL operations can be:

- `Insert: Table Name = schema.name ID = ts,n`
- `Update: Table Name = schema.name ID = ts,n`
- `Delete: Table Name = schema.name ID = ts,n`
- `Insert: Hierarchy Table Name = schema.name ID = ts,n`
- `Update: Hierarchy Table Name = schema.name ID = ts,n`
- `Delete: Hierarchy Table Name = schema.name ID = ts,n`
- `Insert: Summary Table Name = schema.name ID = ts,n`
- `Update: Summary Table Name = schema.name ID = ts,n`
- `Delete: Summary Table Name = schema.name ID = ts,n`
- `Insert: Global Temporary Table ID = ts, tn`
- `Update: Global Temporary Table ID = ts, tn`
- `Delete: Global Temporary Table ID = ts, tn`

Row Identifier (RID) Preparation

For some access plans, it is more efficient if the qualifying row identifiers (RIDs) are sorted and duplicates removed (in the case of index ORing) or that a technique is used to identify RIDs appearing in all indexes being accessed (in the case of index ANDing) before the actual table access is performed. There are three main uses of RID preparation as indicated by the explain statements:

- The following statement indicates that "Index ORing" is used to prepare the list of qualifying RIDs:

  ```
  Index ORing RID Preparation
  ```

  Index ORing refers to the technique of making more than one index access and combining the results to include the distinct RIDs that appear in any of the indexes accessed. The optimizer will consider index ORing when predicates are connected by OR keywords or there is an IN predicate. The index accesses can be on the same index or different indexes.
- Another use of RID preparation is to prepare the input data to be used during list prefetch, as indicated by the following:

  ```
  List Prefetch RID Preparation
  ```
- *Index ANDing* refers to the technique of making more than one index access and combining the results to include RIDs that appear in all of the indexes accessed. Index ANDing processing is started with the statement:

  ```
  Index ANDing
  ```

  If the optimizer has estimated the size of the result set, the estimate is shown with the following statement:

  ```
  Optimizer Estimate of Set Size: n
  ```

  Index ANDing filter operations process RIDs and use bit filter techniques to determine the RIDs which appear in every index accessed. The following statements indicate that RIDs are being processed for index ANDing:

  ```
  Index ANDing Bitmap Build
  Index ANDing Bitmap Probe
  Index ANDing Bitmap Build and Probe
  ```

  If the optimizer has estimated the size of the result set for a bitmap, the estimate is shown with the following statement:

  ```
  Optimizer Estimate of Set Size: n
  ```

For any type of RID preparation, if list prefect can be performed it will be indicated with the statement:

```
Prefetch: Enabled
```

Aggregation

Aggregation is performed on those rows meeting the specified criteria, if any, provided by the SQL statement predicates. If some sort of aggregate function is to be done, one of the following statements appears:

```
Aggregation
Predicate Aggregation
Partial Aggregation
Partial Predicate Aggregation
Intermediate Aggregation
Intermediate Predicate Aggregation
Final Aggregation
Final Predicate Aggregation
```

Predicate aggregation states that the aggregation operation has been pushed-down to be processed as a predicate when the data is actually accessed.

Beneath either of the above aggregation statements will be a indication of the type of aggregate function being performed:

- `Group By`
- `Column Function(s)`
- `Single Record`.

The specific column function can be derived from the original SQL statement. A single record is fetched from an index to satisfy a MIN or MAX operation.

If predicate aggregation is used, then subsequent to the table access statement in which the aggregation appeared, there will be an aggregation ″completion″, which carries out any needed processing on completion of each group or on end-of-file. One of the following lines is displayed:

```
Aggregation Completion
Partial Aggregation Completion
Intermediate Aggregation Completion
Final Aggregation Completion
```

Parallel Processing

Executing an SQL statement in parallel (using either intra-partition or inter-partition parallelism) requires some special operations. The operations for parallel plans are described below.

- When running an intra-partition parallel plan, portions of the plan will be executed simultaneously using several subagents. The creation of the subagents is indicated by the statement:

  ```
  Process Using n Subagents
  ```

- When running an inter-partition parallel plan, the section is broken into several subsections. Each subsection is sent to one or more nodes to be run. An important subsection is the *coordinator subsection*. The coordinator subsection is the first subsection in every plan. It gets control first and is responsible for distributing the other subsections and returning results to the calling application.

The distribution of subsections is indicated by the statement:

```
Distribute Subsection #n
```

The nodes that receive a subsection can be determined in one of eight ways:

– The following indicates that the subsection will be sent to a node within the nodegroup based on the value of the columns.

```
Directed by Hash
|  #Columns = n
|  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
```

– The following indicates that the subsection will be sent to a predetermined node. (This is frequently seen when the statement uses the NODENUMBER() function.)

```
Directed by Node Number
```

– The following indicates that the subsection will be sent to the node corresponding to a predetermined partition number in the given nodegroup. (This is frequently seen when the statement uses the PARTITION() function.)

```
Directed by Partition Number
|  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
```

– The following indicates that the subsection will be sent to the node that provided the current row for the application's cursor.

```
Directed by Position
```

– The following indicates that only one node, determined when the statement was compiled, will receive the subsection.

```
Directed to Single Node
|  Node Number = n
```

– The following indicates that the subsection will be executed on the coordinator node.

```
Directed to Coordinator Node
```

– The following indicates that the subsection will be sent to all the nodes listed.

```
Broadcast to Node List
|  Nodes = n1, n2, n3, ...
```

– The following indicates that only one node, determined as the statement is executing, will receive the subsection.

```
Directed to Any Node
```

• Table queues are used to move data between subsections in a partitioned database environment or between subagents in a symmetric multiprocessor (SMP) environment. Table queues are described as follows:

- The following statements indicate that data is being inserted into a table queue:

  ```
  Insert Into Synchronous Table Queue  ID = qn
  Insert Into Asynchronous Table Queue  ID = qn
  Insert Into Synchronous Local Table Queue  ID = qn
  Insert Into Asynchronous Local Table Queue  ID = qn
  ```
- For database partition table queues, the destination for rows inserted into the table queue is described by one of the following:

  ```
  Broadcast to Coordinator Node
  ```

  All rows are sent to the coordinator node.

  ```
  Broadcast to All Nodes of Subsection n
  ```

  All rows are sent to every database partition that the given subsection is running on.

  ```
  Hash to Specific Node
  ```

  Each row is sent to a database partition based on the values in the row.

  ```
  Send to Specific Node
  ```

  Each row is sent to a database partition determined while the statement is executing.

  ```
  Send to Random Node
  ```

  Each row is sent to a random database partition.
- In some situations, a database partition table queue will have to temporarily overflow some rows to a temporary table. This possibility is identified by the statement:

  ```
  Rows Can Overflow to Temporary Table
  ```
- After a table access that contains a push-down operation to insert rows into a table queue, there will be a "completion" statement which handles rows that could not be immediately sent. One of the following lines is displayed:

  ```
  Insert Into Synchronous Table Queue Completion  ID = qn
  Insert Into Asynchronous Table Queue Completion  ID = qn
  Insert Into Synchronous Local Table Queue Completion  ID = qn
  Insert Into Asynchronous Local Table Queue Completion  ID = qn
  ```
- The following statements indicate that data is being retrieved from a table queue:

  ```
  Access Table Queue  ID = qn
  Access Local Table Queue  ID = qn
  ```

  These messages are always followed by an indication of the number of columns being retrieved.

```
#Columns = n
```

– If the table queue sorts the rows at the receiving end, the table queue access will also have one of the following messages:

```
Output Sorted
Output Sorted and Unique
```

These messages are followed by an indication of the number of keys used for the sort operation.

```
#Key Columns = n
```

For each column in the sort key, one of the following is displayed:

```
Key n: (Ascending)
Key n: (Descending)
```

– If predicates will be applied to rows by the receiving end of the table queue, the following message is shown:

```
Residual Predicate(s)
|  #Predicates = n
```

- Some subsections in a partitioned database environment explicitly loop back to the start of the subsection with the statement:

```
Jump Back to Start of Subsection
```

Federated Statement Processing

Executing an SQL statement in a federated database requires the ability to perform portions of the statement on other data sources.

The following indicates that a data source will be accessed:

```
Distributed Subquery #n
|   #Columns = n
```

It is possible to apply predicates to the data returned from the distributed subquery. The number of predicates being applied will be indicated as follows:

```
Residual Predicate(s)
|   #Predicates = n
```

The details for each distributed subquery is provided separately. The options for distributed subqueries are described below:

- The data source for the subquery is shown by one of the following:

```
Server: server_name (type, version)
Server: server_name (type)
Server: server_name
```

- The SQL statement for the subquery is displayed as:

```
Subquery SQL Statement:
statement
```

- The nicknames referenced in the subquery are listed as follows:

```
Nickname Referenced:
Schema.nickname Base = baseschema.basetable
```

- If values are passed from the federated server to the data source before executing the subquery, the number of values will be shown by:

  ```
  #Input Columns: n
  ```
- If values are passed from the data source to the federated server after executing the subquery, the number of values will be shown by:

  ```
  #Output Columns: n
  ```

Miscellaneous Statements

- Sections for data definition language statements will be indicated in the output with the following:

  ```
  DDL Statement
  ```

  No additional explain output is provided for DDL statements.
- Sections for SET statements for the updatable special registers such as **CURRENT EXPLAIN SNAPSHOT** will be indicated in the output with the following:

  ```
  SET Statement
  ```

  No additional explain output is provided for SET statements.
- If the SQL statement contains the DISTINCT clause, the following text may appear in the output:

  ```
  Distinct Filter  #Columns = n
  ```

  where `n` is the number of columns involved in obtaining distinct rows. To retrieve distinct row values, the rows must be ordered so that duplicates can be skipped. This statement will not appear if the database manager does not have to explicitly eliminate duplicates, as in the following cases:
  – A unique index exists and all the columns in the index key are part of the DISTINCT operation
  – Duplicates that can be eliminated during sorting.
- The following statement will appear if the next operation is dependent on a specific record identifier:

  ```
  Positioned Operation
  ```

  This statement would appear for any SQL statement that uses the WHERE CURRENT OF syntax.
- The following statement will appear if there are predicates that must be applied to the result but that could not be applied as part of another operation:

  ```
  Residual Predicate Application
  |  #Predicates = n
  ```

- The following statement will appear if there is a UNION operator in the SQL statement:

  ```
  UNION
  ```

- The following statement will appear if there is an operation in the access plan, whose sole purpose is to produce row values for use by subsequent operations:

  ```
  Table Constructor
  |  n-Row(s)
  ```

  Table constructors can be used for transforming values in a set into a series of rows that are then passed to subsequent operations. When a table constructor is prompted for the next row, the following statement will appear:

  ```
  Access Table Constructor
  ```

- The following statement will appear if there is an operation which is only processed under certain conditions:

  ```
  Conditional Evaluation
  |  Condition #n:
  |  |  #Predicates = n
  |  Action #n:
  ```

  Conditional evaluation is used to implement such activities as the SQL CASE statement or internal mechanisms such as referential integrity constraints or triggers. If no action is shown, then only data manipulation operations are processed when the condition is true.

- One of the following statements will appear if an ALL, ANY, or EXISTS subquery is being processed in the access plan:
  - `ANY/ALL Subquery`
  - `EXISTS Subquery`
  - `EXISTS SINGLE Subquery`

- Prior to certain UPDATE and DELETE operations, it is necessary to establish the position of a specific row within the table. This is indicated by the following statement:

  ```
  Establish Row Position
  ```

- The following statement will appear if there are rows being returned to the application:

  ```
  Return Data to Application
  |  #Columns = n
  ```

  If the operation was pushed-down into a table access, it will require a completion phase. This phase appears as:

  ```
  Return Data Completion
  ```

Examples of db2expln and dynexpln Output

Five examples are shown here to help understand the layout and format of the output from `db2expln` and `dynexpln`. These examples were run against the SAMPLE database as provided with DB2. A brief discussion is provided for each example. Significant differences from one example to the next have been shown in **bold**.

Example One: No Parallelism Plan

This example is simply requesting a list of all employee names, their jobs, department name and location, and the project name(s) on which they are working. The essence of this access plan is that merge joins are used to join the relevant data from each of the specified tables. Since no indexes are available, the access plan does a relation scan of each table, and each table must be sorted before it can be joined.

```
******************** PACKAGE ***************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:47:58

Bind Timestamp = 2000-01-03-15.47.58.607455

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = No

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno


Estimated Cost        = 126
Estimated Cardinality = 153

Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
|  Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 3
|  |  #Sort Key Columns = 1
|  |  |  Key 1: DEPTNO (Ascending)
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 40
|  |  |  Row Width = 48
|  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
Merge Join
|  Access Table Name = DOOLE.PROJECT  ID = 2,7
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Temp Table  ID = t2
|  |  |  #Columns = 2
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: DEPTNO (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 38
|  |  |  |  Row Width = 28
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
Merge Join
|  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Temp Table  ID = t3
|  |  |  #Columns = 3
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: WORKDEPT (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 63
|  |  |  |  Row Width = 32
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t3
|  Access Temp Table  ID = t3
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
Return Data to Application
|  #Columns = 5

End of section


Optimizer Plan:

               RETURN
               (   1)
                 |
               MSJOIN
               (   2)
              /      \
        MSJOIN        TBSCAN
        (   3)        (  12)
       /      \         |
   TBSCAN    TBSCAN     SORT
   (   4)    (   8)   (  13)
     |         |        |
    SORT      SORT     TBSCAN
   (   5)    (   9)   (  14)
     |         |        |
   TBSCAN    TBSCAN   Table:
   (   6)    (  10)   DOOLE
     |         |      EMPLOYEE
 Table:      Table:
 DOOLE       DOOLE
 DEPARTMENT  PROJECT
```

The first part of the plan accesses the DEPARTMENT and PROJECT tables and uses a merge join to join them. The result of this join is joined to the EMPLOYEE table. The resulting rows are returned to the application.

Example Two: Single-Partition Database Plan with Intra-Partition Parallelism

This example shows the same SQL statement as "Example One: No Parallelism Plan" on page 559, but this query has been compiled for a 4-way SMP machine.

```
******************** PACKAGE ***************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:48:51

Bind Timestamp = 2000-01-03-15.48.51.402403

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
```

```
Intra-Partition Parallel = Yes (Bind Degree = 4)

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 142
Estimated Cardinality = 153

Process Using 4 Subagents
|  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
|  |  #Columns = 3
|  |  Parallel Scan
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Shared Temp Table  ID = t1
|  |  |  #Columns = 3
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: DEPTNO (Ascending)
|  |  |  Use Round-Robin Sort
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 40
|  |  |  |  Row Width = 48
|  |  |  Piped
|  Sorted Shared Temp Table Completion  ID = t1
|  Access Temp Table  ID = t1
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Merge Join
|  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
|  |  |  #Columns = 2
|  |  |  Parallel Scan
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  |  |  Insert Into Sorted Shared Temp Table  ID = t2
|  |  |  |  #Columns = 2
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  Key 1: DEPTNO (Ascending)
|  |  |  |  Use Replicated Sort
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 38
|  |  |  |  |  Row Width = 28
|  |  |  |  Piped
|  |  Sorted Shared Temp Table Completion  ID = t2
|  |  Access Temp Table  ID = t2
|  |  |  #Columns = 2
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  Insert Into Sorted Shared Temp Table  ID = t3
|  |  #Columns = 5
|  |  #Sort Key Columns = 1
|  |  |  Key 1: (Ascending)
|  |  Use Partitioned Sort
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 61
|  |  |  Row Width = 72
|  |  Piped
|  Access Temp Table  ID = t3
|  |  #Columns = 5
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Merge Join
|  |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
|  |  |  #Columns = 3
|  |  |  Parallel Scan
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  |  |  Insert Into Sorted Shared Temp Table  ID = t4
|  |  |  |  #Columns = 3
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  Key 1: WORKDEPT (Ascending)
|  |  |  |  Use Partitioned Sort
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 63
|  |  |  |  |  Row Width = 32
|  |  |  |  Piped
|  |  Sorted Shared Temp Table Completion  ID = t4
|  |  Access Temp Table  ID = t4
|  |  |  #Columns = 3
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 5
Return Data to Application
|  #Columns = 5

End of section
```

```
Optimizer Plan:

               RETURN
               (   1)
                 |
                LTQ
               (   2)
                 |
               MSJOIN
               (   3)
              /      \
        TBSCAN        TBSCAN
        (   4)        (  15)
          |             |
         SORT          SORT
        (   5)        (  16)
          |             |
        MSJOIN        TBSCAN
        (   6)        (  17)
       /      \         |
   TBSCAN    TBSCAN  Table:
   (   7)    (  11)  DOOLE
     |         |     EMPLOYEE
    SORT      SORT
   (   8)    (  12)
     |         |
   TBSCAN    TBSCAN
   (   9)    (  13)
     |         |
 Table:      Table:
 DOOLE       DOOLE
 DEPARTMENT  PROJECT
```

This plan is almost identical to the plan in the first example. The main differences are the creation of four subagents when the plan first starts and the table queue at the end of the plan to gather the results of each of subagent's work before returning them to the application.

It is also interesting to note that an extra sort is needed before joining with EMPLOYEE. This is necessary because the subagents processing the merge join between DEPARTMENT and PROJECT may produce the joined rows out of sequence.

Example Three: Multipartition Database Plan with Inter-Partition Parallelism

This example shows the same SQL statement as "Example One: No Parallelism Plan" on page 559, but this query has been compiled on a partitioned database made up of three database partitions.

```
******************** PACKAGE ***************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:21:29

Bind Timestamp = 2000-01-03-15.21.29.990983

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = Yes
Intra-Partition Parallel = No

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno


Estimated Cost        = 118
Estimated Cardinality = 263

Coordinator Subsection:
   Distribute Subsection #2
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #3
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #1
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Access Table Queue  ID = q1  #Columns = 5
   Return Data to Application
   |  #Columns = 5

Subsection #1:
   Access Table Queue  ID = q2  #Columns = 3
   |  Output Sorted
   |  |  #Key Columns = 1
   |  |  |  Key 1: (Ascending)
   Merge Join
   |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
   |  |  #Columns = 3
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  |  Lock Intents
   |  |  |  Table: Intent Share
   |  |  |  Row  : Next Key Share
   |  |  Insert Into Sorted Temp Table  ID = t1
   |  |  |  #Columns = 3
   |  |  |  #Sort Key Columns = 1
   |  |  |  |  Key 1: DEPTNO (Ascending)
   |  |  |  Sortheap Allocation Parameters:
   |  |  |  |  #Rows     = 40
   |  |  |  |  Row Width = 48
   |  |  |  Piped
   |  Sorted Temp Table Completion  ID = t1
   |  Access Temp Table  ID = t1
   |  |  #Columns = 3
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   Merge Join
   |  Access Table Queue  ID = q3  #Columns = 2
   |  |  Output Sorted
   |  |  |  #Key Columns = 1
   |  |  |  |  Key 1: (Ascending)
   Insert Into Asynchronous Table Queue  ID = q1
   |  Broadcast to Coordinator Node
   |  Rows Can Overflow to Temporary Table

Subsection #2:
   Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
   |  #Columns = 3
   |  Relation Scan
   |  |  Prefetch: Eligible
   |  Lock Intents
   |  |  Table: Intent Share
   |  |  Row  : Next Key Share
   |  Insert Into Sorted Temp Table  ID = t2
   |  |  #Columns = 3
   |  |  #Sort Key Columns = 1
   |  |  |  Key 1: WORKDEPT (Ascending)
   |  |  Sortheap Allocation Parameters:
   |  |  |  #Rows     = 27
   |  |  |  Row Width = 32
   |  |  Piped
   Sorted Temp Table Completion  ID = t2
   Access Temp Table  ID = t2
   |  #Columns = 3
   |  Relation Scan
   |  |  Prefetch: Eligible
   |  Insert Into Asynchronous Table Queue  ID = q2
   |  |  Hash to Specific Node
   |  |  Rows Can Overflow to Temporary Tables
   Insert Into Asynchronous Table Queue Completion  ID = q2

Subsection #3:
   Access Table Name = DOOLE.PROJECT  ID = 2,7
   |  #Columns = 2
   |  Relation Scan
   |  |  Prefetch: Eligible
   |  Lock Intents
   |  |  Table: Intent Share
```

```
   |  |  Row  : Next Key Share
   |  Insert Into Sorted Temp Table  ID = t3
   |  |  #Columns = 2
   |  |  #Sort Key Columns = 1
   |  |  |  Key 1: DEPTNO (Ascending)
   |  |  Sortheap Allocation Parameters:
   |  |  |  #Rows     = 38
   |  |  |  Row Width = 28
   |  |  Piped
   Sorted Temp Table Completion  ID = t3
   Access Temp Table  ID = t3
   |  #Columns = 2
   |  Relation Scan
   |  |  Prefetch: Eligible
   |  Insert Into Asynchronous Table Queue  ID = q3
   |  |  Hash to Specific Node
   |  |  Rows Can Overflow to Temporary Tables
   Insert Into Asynchronous Table Queue Completion  ID = q3

End of section


Optimizer Plan:

               RETURN
               (   1)
                 |
                BTQ
               (   2)
                 |
               MSJOIN
               (   3)
              /      \
       MSJOIN         MDTQ
       (   4)        (  14)
      /      \          |
   MDTQ     TBSCAN    TBSCAN
  (   5)    (  10)    (  15)
    |         |          |
  TBSCAN     SORT      SORT
  (   6)    (  11)    (  16)
    |         |          |
   SORT     TBSCAN    TBSCAN
  (   7)    (  12)    (  17)
    |         |          |
  TBSCAN   Table:     Table:
  (   8)   DOOLE      DOOLE
    |      DEPARTMENT PROJECT
 Table:
 DOOLE
 EMPLOYEE
```

This plan has all the same pieces as the plan in the first example, but the section has been broken into four subsections. The subsections have the following tasks:

- **Coordinator Subsection**. This subsection coordinates the other subsections. In this plan, it causes the other subsections to be distributed and then uses a table queue to gather the results to be returned to the application.
- **Subsection #1**. This subsection scans table queue q2 and uses a merge join to join it with the DEPARTMENT table. A second merge join then adds in the data from table queue q3. The joined rows are then sent to the coordinator subsection using table queue q1.
- **Subsection #2**. This subsection scans the EMPLOYEE table, sorts it, and hashes to a specific node with the results. These results are read by Subsection #1.
- **Subsection #3**. This subsection scans the PROJECT table, sorts it, and hashes to a specific node with the results. These results are read by Subsection #1.

Example Four: Multipartition Database Plan with Inter-Partition and Intra-Partition Parallelism

This example shows the same SQL statement as "Example One: No Parallelism Plan" on page 559, but this query has been compiled on a partitioned database made up of three database partitions, each of which is on a four-way SMP machine.

```
******************** PACKAGE ***************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:22:14

Bind Timestamp = 2000-01-03-15.22.14.659970

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = Yes
Intra-Partition Parallel = Yes (Bind Degree = 4)

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
```

```
          = z.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 140
Estimated Cardinality = 263

Coordinator Subsection:
   Distribute Subsection #2
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #3
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #1
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Access Table Queue  ID = q1  #Columns = 5
   Return Data to Application
   |  #Columns = 5

Subsection #1:
   Process Using 4 Subagents
   |  Access Table Queue  ID = q3  #Columns = 3
   |  Insert Into Sorted Shared Temp Table  ID = t1
   |  |  #Columns = 3
   |  |  #Sort Key Columns = 1
   |  |  |  Key 1: (Ascending)
   |  |  Use Partitioned Sort
   |  |  Sortheap Allocation Parameters:
   |  |  |  #Rows     = 27
   |  |  |  Row Width = 32
   |  |  Piped
   |  Access Temp Table  ID = t1
   |  |  #Columns = 3
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  Merge Join
   |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
   |  |  |  #Columns = 3
   |  |  |  Parallel Scan
   |  |  |  Relation Scan
   |  |  |  |  Prefetch: Eligible
   |  |  |  Lock Intents
   |  |  |  |  Table: Intent Share
   |  |  |  |  Row  : Next Key Share
   |  |  |  Insert Into Sorted Shared Temp Table  ID = t2
   |  |  |  |  #Columns = 3
   |  |  |  |  #Sort Key Columns = 1
   |  |  |  |  |  Key 1: DEPTNO (Ascending)
   |  |  |  |  Use Partitioned Sort
   |  |  |  |  Sortheap Allocation Parameters:
   |  |  |  |  |  #Rows     = 40
   |  |  |  |  |  Row Width = 48
   |  |  |  |  Piped
   |  |  Sorted Shared Temp Table Completion  ID = t2
   |  |  Access Temp Table  ID = t2
   |  |  |  #Columns = 3
   |  |  |  Relation Scan
   |  |  |  |  Prefetch: Eligible
   |  Insert Into Sorted Shared Temp Table  ID = t3
   |  |  #Columns = 6
   |  |  #Sort Key Columns = 1
   |  |  |  Key 1: (Ascending)
   |  |  Use Partitioned Sort
   |  |  Sortheap Allocation Parameters:
   |  |  |  #Rows     = 44
   |  |  |  Row Width = 76
   |  |  Piped
   |  Access Temp Table  ID = t3
   |  |  #Columns = 6
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  Merge Join
   |  |  Access Table Queue  ID = q5  #Columns = 2
   |  |  Insert Into Sorted Shared Temp Table  ID = t4
   |  |  |  #Columns = 2
   |  |  |  #Sort Key Columns = 1
   |  |  |  |  Key 1: (Ascending)
   |  |  |  Use Partitioned Sort
   |  |  |  Sortheap Allocation Parameters:
   |  |  |  |  #Rows     = 38
   |  |  |  |  Row Width = 28
   |  |  |  Piped
   |  |  Access Temp Table  ID = t4
   |  |  |  #Columns = 2
   |  |  |  Relation Scan
   |  |  |  |  Prefetch: Eligible
   |  Insert Into Asynchronous Local Table Queue  ID = q2
   Access Local Table Queue  ID = q2  #Columns = 5
   Insert Into Asynchronous Table Queue  ID = q1
   |  Broadcast to Coordinator Node
   |  Rows Can Overflow to Temporary Table

Subsection #2:
   Process Using 4 Subagents
   |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
   |  |  #Columns = 3
   |  |  Parallel Scan
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  |  Lock Intents
   |  |  |  Table: Intent Share
   |  |  |  Row  : Next Key Share
   |  |  Insert Into Sorted Shared Temp Table  ID = t5
   |  |  |  #Columns = 3
   |  |  |  #Sort Key Columns = 1
   |  |  |  |  Key 1: WORKDEPT (Ascending)
   |  |  |  Use Round-Robin Sort
   |  |  |  Sortheap Allocation Parameters:
```

```
|  |  |  |  #Rows     = 27
|  |  |  |  Row Width = 32
|  |  |  Piped
|  Sorted Shared Temp Table Completion  ID = t5
|  Access Temp Table  ID = t5
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q4
Access Local Table Queue  ID = q4  #Columns = 3
Insert Into Asynchronous Table Queue  ID = q3
|  Hash to Specific Node
|  Rows Can Overflow to Temporary Tables

Subsection #3:
   Process Using 4 Subagents
   |  Access Table Name = DOOLE.PROJECT  ID = 2,7
   |  |  #Columns = 2
   |  |  Parallel Scan
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  |  Lock Intents
   |  |  |  Table: Intent Share
   |  |  |  Row  : Next Key Share
   |  |  Insert Into Sorted Shared Temp Table  ID = t6
   |  |  |  #Columns = 2
   |  |  |  #Sort Key Columns = 1
   |  |  |  |  Key 1: DEPTNO (Ascending)
   |  |  |  Use Round-Robin Sort
   |  |  |  Sortheap Allocation Parameters:
   |  |  |  |  #Rows     = 38
   |  |  |  |  Row Width = 28
   |  |  |  Piped
   |  Sorted Shared Temp Table Completion  ID = t6
   |  Access Temp Table  ID = t6
   |  |  #Columns = 2
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  Insert Into Asynchronous Local Table Queue  ID = q6
   Access Local Table Queue  ID = q6  #Columns = 2
   Insert Into Asynchronous Table Queue  ID = q5
   |  Hash to Specific Node
   |  Rows Can Overflow to Temporary Tables

End of section


Optimizer Plan:

              RETURN
              (   1)
                |
               BTQ
              (   2)
                |
               LTQ
              (   3)
                |
              MSJOIN
              (   4)
             /      \
       TBSCAN         TBSCAN
       (   5)         (  20)
         |              |
        SORT           SORT
       (   6)         (  21)
         |              |
       MSJOIN          DTQ
       (   7)         (  22)
      /      \          |
  TBSCAN    TBSCAN     LTQ
  (   8)    (  16)    (  23)
    |         |         |
   SORT      SORT     TBSCAN
  (   9)    (  17)    (  24)
    |         |         |
   DTQ      TBSCAN     SORT
  (  10)    (  18)    (  25)
    |         |         |
   LTQ    Table:      TBSCAN
  (  11)  DOOLE       (  26)
    |     DEPARTMENT    |
  TBSCAN              Table:
  (  12)              DOOLE
    |                 PROJECT
   SORT
  (  13)
    |
  TBSCAN
  (  14)
    |
 Table:
 DOOLE
 EMPLOYEE
```

This plan is similar to that in “Example Three: Multipartition Database Plan with Inter-Partition Parallelism” on page 564, except that multiple subagents execute each subsection. Also, at the end of each subsection, a local table queue gathers the results from all of the subagents before the qualifying rows are inserted into the second table queue to be hashed to a specific node.

Example Five: Federated Database Plan

This example shows the same SQL statement as “Example One: No Parallelism Plan” on page 559, but this query has been compiled on a federated database where the tables DEPARTMENT and PROJECT are on a data source and the table EMPLOYEE is on the federated server.

```
******************** PACKAGE ***************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 16:29:01

Bind Timestamp = 2000-01-03-16.29.01.479230

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = No

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno

Estimated Cost        = 1954
Estimated Cardinality = 100800
```

```
Distribute Subquery #2
   |  #Columns = 3
   Insert Into Sorted Shared Temp Table  ID = t1
   |  #Columns = 3
   |  #Sort Key Columns = 1
   |  |  Key 1: Remote Query #2, Output Column 1 (Ascending)
   |  Sortheap Allocation Parameters:
   |  |  #Rows     = 1000
   |  |  Row Width = 56
   |  Piped
   Access Temp Table  ID = t1
   |  #Columns = 3
   |  Relation Scan
   |  |  Prefetch: Eligible
   Merge Join
   |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,5
   |  #Columns = 3
   |  Relation Scan
   |  |  Prefetch: Eligible
   |  |  Lock Intents
   |  |  |  Table: Intent Share
   |  |  |  Row  : Next Key Share
   |  |  Insert Into Sorted Temp Table  ID = t2
   |  |  |  #Columns = 3
   |  |  |  #Sort Key Columns = 1
   |  |  |  |  Key 1: WORKDEPT (Ascending)
   |  |  |  Sortheap Allocation Parameters:
   |  |  |  |  #Rows     = 63
   |  |  |  |  Row Width = 32
   |  |  |  Piped
   |  Sorted Temp Table Completion  ID = t2
   |  Access Temp Table  ID = t2
   |  |  #Columns = 3
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   Merge Join
   |  Distribute Subquery #1
   |  |  #Columns = 2
   |  Insert Into Sorted Temp Table  ID = t3
   |  |  #Columns = 2
   |  |  |  Key 1: Remote Query #1, Output Column 1 (Ascending)
   |  |  Sortheap Allocation Parameters:
   |  |  |  #Rows     = 1000
   |  |  |  Row Width = 36
   |  |  Piped
   |  Access Temp Table  ID = t3
   |  |  #Columns = 2
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   Return Data to Application
   |  #Columns = 5

   Distributed Subquery #1:
   Server: REMOTE_SAMPLE  (DB2/CS 7.1)
   Subquery SQL Statement:

        SELECT A0."DEPTNO", A0."PROJNAME"
        FROM "DOOLE"."PROJECT" A0

   Nicknames Referenced:
      REMOTE.PROJECT  ID = 7  Base = DOOLE.PROJECT
   #Output Columns = 2

   Distributed Subquery #2:
   Server: REMOTE_SAMPLE  (DB2/CS 7.1)
   Subquery SQL Statement:

        SELECT A0."DEPTNO", A0."DEPTNAME", A0."LOCATION"
        FROM "DOOLE"."DEPARTMENT" A0

   Nicknames Referenced:
      REMOTE.DEPARTMENT  ID = 4  Base = DOOLE.DEPARTMENT
   #Output Columns = 3

End of section


Optimizer Plan:
```

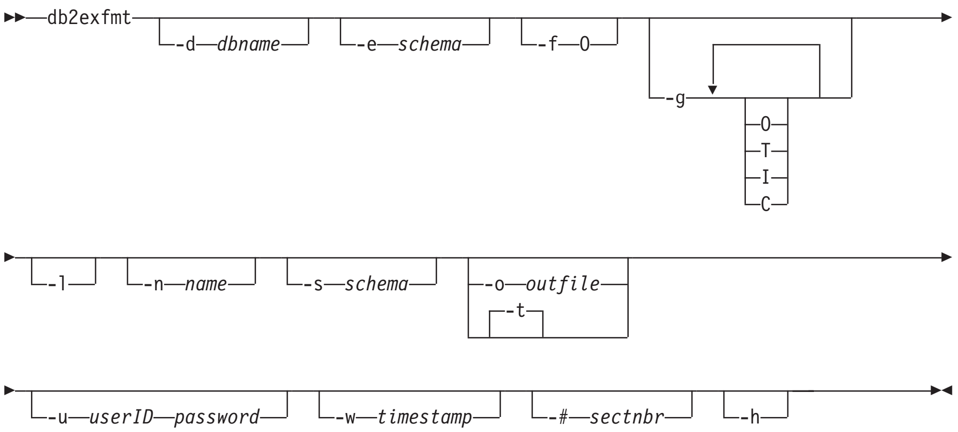
```
                RETURN
                (   1)
                  |
                MSJOIN
                (   2)
              /        \
       MSJOIN            TBSCAN
       (   3)            (  13)
      /      \             |
   TBSCAN     TBSCAN      SORT
   (   4)     (   9)     (  14)
     |          |          |
    SORT       SORT      DSBQRY
   (   5)     (  10)     (  15)
     |          |          |
   DSBQRY     TBSCAN    Nickname:
   (   6)     (  11)    REMOTE
     |          |       PROJECT
  Nickname:   Table:
  REMOTE      DOOLE
  DEPARTMENT  EMPLOYEE
```

This plan has all the same pieces as the plan in the first example, except that the data for two of the tables are coming from data sources. The two tables are accessed through distributed subqueries which, in this case, simply select all the rows from those tables. Once the data is returned to the federated server, it is joined to the data from the local table.

Appendix D. db2exfmt - Explain Table Format Tool

You use the db2exfmt tool to format the contents of the explain tables. This tool is located in the `misc` subdirectory of the instance `sqllib` directory.

To use the tool, you require read access to the explain tables being formatted.



-d dbname
: Name of the database containing packages.

-e schema
: Explain table schema.

-f
: Formatting flags. In this release, the only supported value is `O` (operator summary).

-g
: Graph plan. If only `-g` is specified, a graph, followed by formatted information for all of the tables, is generated. Otherwise, any combination of the following valid values can be specified:

O	Generate a graph only. Do not format the table contents.
T	Include total cost under each operator in the graph.
I	Include I/O cost under each operator in the graph.
C	Include the expected output cardinality (number of tuples) of each operator in the graph.

-l
: Respect case when processing package names.

-n name
: Name of the source of the explain request (SOURCE_NAME).

-s schema
: Schema or qualifier of the source of the explain request (SOURCE_SCHEMA).

-o outfile
Output file name.

-t Direct the output to the terminal.

-u user ID password
When connecting to a database, use the provided user ID and password.

Both the user ID and password must be valid according to naming conventions and be recognized by the database.

-w timestamp
Explain time stamp. Specify `-1` to obtain the latest explain request.

-# sectnbr
Section number in the source. To request all sections, specify zero.

-h Display help information. When this option is specified, all other options are ignored, and only the help information is displayed.

You will be prompted for any parameter values that are not supplied, or that are incompletely specified, except in the case of the `-h` and the `-l` options.

If an explain table schema is not provided, the value of the environment variable **USER** is used as the default. If this variable is not found, the user is prompted for an explain table schema.

Source name, source schema, and explain time stamp can be supplied in LIKE predicate form, which allows the percent sign (%) and the underscore (_) to be used as pattern matching characters to select multiple sources with one invocation. For the latest explained statement, the explain time can be specified as `-1`.

If `-o` is specified without a file name, and `-t` is not specified, the user is prompted for a file name (the default name is `db2exfmt.out`). If neither `-o` nor `-t` is specified, the user is prompted for a file name (the default option is terminal output). If `-o` and `-t` are both specified, the output is directed to the terminal.

Appendix E. Configuring XA Transaction Managers to Use DB2 UDB

The sections that follow describe how to configure specific products to use DB2 as a resource manager. You can use any of the following:

- "Configuring IBM TXSeries CICS"
- "Configuring IBM TXSeries Encina"
- "Configuring BEA Tuxedo" on page 582
- "Configuring Microsoft Transaction Server" on page 584.

Configuring IBM TXSeries CICS

For information about how to configure IBM TXSeries CICS to use DB2 as a resource manager, refer to your *IBM TXSeries CICS Administration Guide*. TXSeries documentation can be viewed online at starting at http://www.transarc.com/dfs/public/www/htdocs/.hosts/external/Library/index.html

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

Configuring IBM TXSeries Encina

The following are the various API and configuration parameters required for the integration of Encina Monitor and DB2 Universal Database servers or DB2 for MVS, DB2 for OS/390, DB2 for AS/400, or DB2 for VSE&VM when accessed via DB2 Connect. TXSeries documentation can be viewed online starting at
http://www.transarc.com/dfs/public/www/htdocs/.hosts/external/Library/index.html

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

Configuring DB2

To configure DB2:

1. Each database name must be defined in the DB2 database directory. If the database is a remote database, then a Node Directory entry must also be defined. You can perform the configuration using the GUI Client Configuration Assistant (CCA), or the DB2 Command Line Processor (CLP). For example:

```
DB2 CATALOG DATABASE inventdb AS inventdb AT NODE host1 AUTH SERVER

DB2 CATALOG TCPIP NODE host1 REMOTE hostname1 SERVER svcname1
```

2. The DB2 client can optimize its internal processing for Encina if it knows that it is dealing with Encina. You can specify this by setting the *tp_mon_name* database manager configuration parameter to ENCINA. The default is for no special optimization. If *tp_mon_name* is set, then the application must ensure the thread that performs the unit of work also immediately commits the work after ending it. No other unit of work may be started. If this is *not* your environment, then ensure that the value for *tp_mon_name* value is NONE (or via the CLP, the value is set to NULL). The *tp_mon_name* can be updated by invoking the CCA or by the CLP:
   - On AIX use:
     ```
     UPDATE DATABASE MANAGER CONFIGURATION USING TP_MON_NAME ENCINA
     ```
   - On Windows NT use:
     ```
     UPDATE DATABASE MANAGER CONFIGURATION USING TP_MON_NAME
     libEncServer:E
     ```
     In Intel environments, this parameter contains the path and name of the DLL in an external transaction manager product containing the functions ax_reg and ax_unreg, and also informs DB2 which TP Monitor is being used.

Configuring Encina for Each Resource Manager

To configure Encina for each resource manager, an administrator must define the Open String, Close String, and Thread of Control Agreement for each DB2 database as a resource manager before the resource manager can be registered for transactions in an application. The configuration can be performed using the Enconcole full screen interface, or the Encina command line interface. For example:

```
monadmin create rm inventdb  -open "inventdb,user1,password1"
```

There is one resource manager configuration for each DB2 database, and each resource manager (RM) configuration must have an rm name (″logical RM name″). To simplify the situation, you should make it identical to the database name.

The XA Open String contains information that is required to establish a connection to the database. The content of the string is RM specific. The XA Open String of DB2 UDB contains the alias name of the database to be opened, and optionally a userID and password to be associated with the connection. Note that the database name defined here must also be cataloged into the regular database directory required for all database access. The name can be up to 8 bytes long.

The XA Close String is not used by DB2.

The Thread of Control Agreement determines if an application agent thread can handle more than one transaction at a time. DB2 V5.0 and following supports the default of TMXA_SERIALIZE_ALL_OPERATIONS, where a thread can be reused only after a transaction has completed.

If you are accessing DB2 for OS/390, DB2 for MVS, DB2 for AS/400, or DB2 for VSE&VM, then you must use the DB2 Syncpoint Manager. Please refer to the *DB2 Connect Enterprise Edition for OS/2 and Windows Quick Beginnings* manual for configuration instructions.

Referencing a DB2 Database from an Encina Application

To reference a DB2 database from an Encina application:

1. Use the Encina Scheduling Policy API to specify how many application agents can be run from a single TP Monitor application process. For example:

   ```
   rc = mon_SetSchedulingPolicy (MON_EXCLUSIVE)
   ```

   For DB2 (DB2 Universal Database, host, or AS/400 database servers), you should use the default setting of MON_EXCLUSIVE. This ensures that:

   - The application process is locked during the life time of the transaction.
   - The application acts single threaded.

   Note: If you are using the ODBC or DB2 Call Level Interface, you must disable the multithread support. You can do this by setting the CLI configuration parameter `DISABLEMULTITHREAD = 1` (disables multithreading). The default for DB2 Universal Database is `DISABLEMULTITHREAD = 0` (enables multithreading). Refer to the *CLI Guide and Reference* for more information.

2. Use the Encina RM Registration API to provide the XA switch and the logical RM name to be used by Encina when referencing the RM in an application process. For example:

   ```
   rc = mon_RegisterRmi ( &db2xa_switch, /* xa switch */
                            "inventdb", /* logical RM name */
                          &rmiId  ); /* internal RM id */
   ```

   The XA Switch contains the addresses of the XA routines in the RM that the TM can call, and it also specifies the functionality that is provided by the RM. The XA Switch of DB2 Universal Database is `db2xa_switch`, and it resides in the DB2 client library (`db2app.dll` on INTEL platforms and `libdb2` on UNIX-based platforms).

   The logical RM name is the one used by Encina, and is not the actual database name that is used by the SQL application that runs under Encina. The actual database name is specified in the XA Open String in the Encina RM Registration. To simplify the situation, the logical RM name is set to be the same as the database name in this example.

The third parameter returns an internal identifier or handle that is used by the TM to reference this connection.

Note: When using Encina for transaction processing with DB2 through the TM-XA interface, note that Encina nested transactions are not currently supported by the DB2 XA interface. If possible, avoid using these transactions. If you cannot, ensure that SQL work is done in only one member of the Encina transaction family.

Configuring BEA Tuxedo

Note: Applications that access host or AS/400 database servers in a Tuxedo environment are limited to read-only access to these servers.

To configure Tuxedo to use DB2 as a resource manager, perform the following steps:

1. Install Tuxedo as specified in the documentation for that product. Ensure that you perform all basic Tuxedo configuration, including the log files and environment variables.

   You also require a compiler and the DB2 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. Also set the PATH variable to include the DB2 program directories. Then confirm that the Tuxedo server ID can connect to the DB2 databases.
3. For Windows NT only. Update the *tp_mon_name* database manager configuration parameter with the name of the DLL that contains the `ax_reg` and `ax_unreg` routines. In Tuxedo, this DLL is called `libtux`.
4. Add a definition for DB2 to the Tuxedo resource manager definition file. In the examples that follow, `UDB_XA` is the locally defined Tuxedo resource manager name for DB2, and `db2xa_switch` is the DB2-defined name for a structure of type `xa_switch_t`:
   - For AIX. In the file `${TUXDIR}/udataobj/RM`, add the definition:
     ```
     # DB2 UDB
     UDB_XA:db2xa_switch:-L${DB2DIR} /lib -ldb2
     ```
     Where `{TUXDIR}` is the directory where you installed Tuxedo, and `{DB2DIR}` is the DB2 instance directory.
   - For Windows NT. In the file `%TUXDIR%\udataobj\rm`, add the definition:
     ```
     # DB2 UDB
     UDB_XA;db2xa_switch;%DB2DIR%\lib\db2api.lib
     ```

Where `%TUXDIR%` is the directory where you installed Tuxedo, and `%DB2DIR%` is the DB2 instance directory.

5. Build the Tuxedo transaction monitor server program for DB2:
   - For AIX:
     ```
     ${TUXDIR}/bin/buildtms -r UDB_XA -o ${TUXDIR}/bin/TMS_UDB
     ```
     Where `{TUXDIR}` is the directory where you installed Tuxedo.
   - For Windows NT:
     ```
     %TUXDIR%\bin\buildtms -r UDB_XA -o %TUXDIR%\bin\TMS_UDB
     ```
6. Build the application servers. In the examples that follow, the `-r` option specifies the resource manager name, the `-f` option (used one or more times) specifies the files that contain the application services, the `-s` option specifies the application service names for this server, and the `-o` option specifies the output server file name:
   - For AIX:
     ```
     ${TUXDIR}/bin/buildserver -r UDB_XA -f svcfile.o -s SVC1,SVC2
        -o UDBserver
     ```
     Where `{TUXDIR}` is the directory where you installed Tuxedo.
   - For Windows NT:
     ```
     %TUXDIR%\bin\buildserver -r UDB_XA -f svcfile.o -s SVC1,SVC2
        -o UDBserver
     ```
     Where `%TUXDIR%` is the directory where you installed Tuxedo.
7. Set up the Tuxedo configuration file to reference the DB2 server. In the *GROUPS section of the `UDBCONFIG` file, add an entry similar to:
   ```
   UDB_GRP   LMID=simp GRPNO=3
     TMSNAME=TMS_UDB TMSCOUNT=2
     OPENINFO="UDB_XA:SAMPLE,db2_user,,db2_user_pwd"
   ```
   Where the TMSNAME parameter specifies the transaction monitor server program that you built previously, and the OPENINFO parameter specifies the resource manager name. This is followed by the database name and the DB2 user and password, which are used for authentication.

   The application servers that you built previously are referenced in the *SERVERS section of the Tuxedo configuration file.
8. Start Tuxedo:
   ```
   tmboot -y
   ```
   After the command completes, Tuxedo messages should indicate that the servers are started. In addition, if you issue the DB2 command LIST

APPLICATIONS ALL, you should see two connections (in this situation, specified by the TMSCOUNT parameter in the UDB group in the Tuxedo configuration file, `UDBCONFIG`.

Configuring Microsoft Transaction Server

DB2 UDB V5.2 and later can be fully integrated with Microsoft Transaction Server (MTS) Version 2.0. Applications running under MTS on Windows 32-bit operating systems can use MTS to coordinate two-phase commit with multiple DB2 UDB, host, and AS/400 database servers, as well as with other MTS-compliant resource managers.

Enabling MTS Support in DB2

Microsoft Transaction Server support is automatically enabled. While you can set the *tp_mon_name* database manager configuration parameter to `MTS`, it is not necessary and will be ignored.

Note: Additional technical information may be provided on the IBM web site to assist you with installation and configuration of DB2 MTS support. Set your URL to http://www.ibm.com/software/data/db2/library/, and search for a DB2 Universal Database ″Technote″ with the keyword ″MTS″.

MTS Software Prerequisites

MTS support requires the DB2 Client Application Enabler (CAE) Version 5.2, or later, and MTS must be at Version 2.0 with Hotfix 0772 or later releases.

The installation of the DB2 ODBC driver on Windows 32-bit operating systems will automatically add a new keyword into the registry:

```
HKEY_LOCAL_MACHINE\software\ODBC\odbcinit.ini\IBM DB2 ODBC Driver:
Keyword Value Name: CPTimeout
Data Type: REG_SZ
Value: 60
```

Installation and Configuration

Following is a summary of installation and configuration considerations for MTS. To use DB2’s MTS support, you must:

1. Install MTS and the DB2 client on the same machine where the MTS application runs.
2. If host or AS/400 database servers are to be involved in a multisite update:
   a. Install DB2 Connect Enterprise Edition (EE), either on your local machine or on a remote machine. DB2 Connect EE allows host or AS/400 database servers to participate in a multisite update transaction.

b. Ensure that your DB2 Connect EE server is enabled for multisite update. For information about enabling DB2 Connect for multisite updates, refer to the DB2 Connect Enterprise Edition Quick Beginnings manual for your platform.

When running DB2 CLI/ODBC applications, the following configuration keywords (as set in the `db2cli.ini` file) must not be changed from their default values:

- CONNECTYPE keyword (default 1)
- MULTICONNECT keyword (default 1)
- DISABLEMULTITHREAD keyword (default 1)
- CONNECTIONPOOLING keyword (default 0)
- KEEPCONNECTION keyword (default 0)

DB2 CLI applications written to make use of MTS support must not change the attribute values corresponding to the above keywords. In addition, the applications must not change the default values of the following attributes:

- SQL_ATTR_CONNECT_TYPE attribute (default SQL_CONCURRENT_TRANS)
- SQL_ATTR_CONNECTON_POOLING attribute (default SQL_CP_OFF)

Note: Additional technical information may be provided on the IBM web site to assist you with installation and configuration of DB2 MTS support. Set your URL to http://www.ibm.com/software/data/db2/library/, and search for a DB2 Universal Database ″Technote″ with the keyword ″MTS″.

Verifying the Installation

1. Configure your DB2 client and DB2 Connect EE to access your DB2 UDB, host, or AS/400 server.
2. Verify the connection from the DB2 CAE machine to the DB2 UDB database servers.
3. Verify the connection from the DB2 Connect machine to your host or AS/400 database server with DB2 CLP, and issue a few queries.
4. Verify the connection from the DB2 CAE machine through the DB2 Connect gateway to your host or AS/400 database server, and issue a few queries.

Supported DB2 Database Servers

The following servers are supported for multisite update using MTS-coordinated transactions:

- DB2 Universal Database Enterprise Edition Version 5.2
- DB2 Enterprise - Extended Edition Version 5.2
- DB2 for OS/390

- DB2 for MVS
- DB2 for AS/400
- DB2 for VM&VSE
- DB2 Common Server for SCO, Version 2
- DB2 Universal Database for AIX with PTF U453782
- DB2 Universal Database for HP-UX with PTF U453784
- DB2 Universal Database Enterprise Edition for OS/2 with PTF WR09033
- DB2 Universal Database for SOLARIS with PTF U453783
- DB2 Universal Database Enterprise Edition for Windows NT with PTF WR09034
- DB2 Universal Database Extended Enterprise Edition for UNIX or Windows NT.

MTS Transaction Time-Out and DB2 Connection Behavior

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

If a transaction takes longer than the transaction time-out value (default value is 60 seconds), MTS will asynchronously issue an abort to all Resource Managers involved, and the whole transaction is aborted.

For the connection to a DB2 server, the abort is translated into a DB2 rollback request. Like any other database request, the rollback request is serialized on the connection to guarantee the integrity of the data on the database server.

As a result:

- If the connection is idle, the rollback is executed immediately.
- If a long-running SQL statement is processing, the rollback request waits until the SQL statement finishes.

Connection Pooling

Connection pooling enables an application to use a connection from a pool of connections, so that the connection does not need to be re-established for each use. Once a connection has been created and placed in a pool, an application can reuse that connection without performing a complete connection process. The connection is pooled when the application disconnects from the ODBC data source, and will be given to a new connection whose attributes are the same.

Connection pooling has been a feature of ODBC driver Manager 2.x. With the latest ODBC driver manager (version 3.5) that was shipped with MTS, connection pooling has some configuration changes and new behavior for

ODBC connections of transactional MTS COM objects (see “Reusing ODBC Connections Between COM Objects Participating in the Same Transaction” on page 588).

ODBC driver Manager 3.5 requires that the ODBC driver register a new keyword in the registry before it allows connection pooling to be activated. The keyword is:

```
  Key Name: SOFTWARE\ODBC\ODBCINST.INI\IBM DB2 ODBC DRIVER
  Name: CPTimeout
  Type: REG_SZ
  Data: 60
```

The DB2 ODBC driver Version 6 and later for the 32-bit Windows operating system fully supports connection pooling; therefore, this keyword is registered. Version 5.2 clients must install FixPack 3 (WR09024) or later.

The default value of 60 means that the connection will be pooled for 60 seconds before it is disconnected.

In a busy environment, it is better to increase the CPTimeout value to a large number (Microsoft sometimes suggests 10 minutes for certain environments) to prevent too many physical connects and disconnects, because these consume large amounts of system resource, including system memory and communications stack resources.

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

```
REGEDIT4

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

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

MTS Connection Pooling using ADO 2.1 and Later

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

```
REGEDIT4

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

Reusing ODBC Connections Between COM Objects Participating in the Same Transaction

ODBC connections in MTS COM objects have connection pooling turned on automatically (whether or not the COM object is transactional).

For multiple MTS COM objects participating in the same transaction, the connection can be reused between two or more COM objects in the following manner.

Suppose that there are two COM objects, COM1 and COM2, that connect to the same ODBC data source and participate in the same transaction.

After COM1 connects and does its work, it disconnects, and the connection is pooled. However, this connection will be reserved for the use of other COM objects of the same transaction. It will be available to other transactions only after the current transaction ends.

When COM2 is invoked in the same transaction, it is given the pooled connection. MTS will ensure that the connection can only be given to the COM objects that are participating in the same transaction.

On the other hand, if COM1 does not explicitly disconnect, it will tie up the connection until the transaction ends. When COM2 is invoked in the same transaction, a separate connection will be acquired. Subsequently, this transaction ties up two connections instead of one.

This reuse of connection feature for COM objects participating in the same transaction is preferable for the following reasons:

- It uses fewer resources in both the client and the server. Only one connection is needed.
- It eliminates the possibility that two connections participating in the same transaction (accessing the same database server and accessing the same data) can lock one another, because DB2 servers treat different connections from MTS COM objects as separate transactions.

Tuning TCP/IP Communications

If a small CPTimeout value is used in a high-workload environment where too many physical connects and disconnects occur at the same time, the TCP/IP stack may encounter resource constraints.

To alleviate this problem, use the TCP/IP Registry Entries. These are described in the *Windows NT Resource Guide*, Volume 1. The registry key values are located in HKEY_LOCAL_MACHINE—> SYSTEM—> CurrentControlSet—> Services—> TCPIP—> Parameters.

The default values and suggested settings are as follows:

Name	**Default Value**	**Suggested Value**
KeepAlive time	7200000 (2 hours)	Same
KeepAlive interval	1000 (1 second)	10000 (10 seconds)
TcpKeepCnt	120 (2 minutes)	240 (4 minutes)
TcpKeepTries	20 (20 re-tries)	Same
TcpMaxConnectAttempts	3	6
TcpMaxConnectRetransmission	3	6
TcpMaxDataRetransmission	5	8
TcpMaxRetransmissionAttempts	7	10
If the registry value is not defined, create it.		

Testing DB2 With The MTS ″BANK″ Sample Application

You can use the ″BANK″ sample program that is shipped with MTS to test the setup of the client products and MTS.

Follow these steps:

1. Change the file `\Program Files\Common Files\ODBC\Data Sources\MTSSamples.dsn` so that it looks like this:

   ```
   [ODBC]
   DRIVER=IBM DB2 ODBC DRIVER
   UID=your_user_id
   PWD=your_password
   DSN=your_database_alias
   Description=MTS Samples
   ```

   where:

   - *your_user_id*and *your_password* are the user ID and password used to connect to the host.
   - *your_database_alias* is the database alias used to connect to the database server.
2. Go to ODBC Administration in the Control Panel, select the **System DSN** tab, and then add the data source:
   a. Select IBM ODBC Driver, and then select **Finish**.

   b. When presented with the list of database aliases, choose the one that was specified previously.
   c. Select **OK**.
3. Use DB2 CLP to connect to a DB2 database under the ID *your_user_id*, as above.
   a. Bind the `db2cli.lst` file:
      ```
      db2 bind @C:\sqllib\bnd\db2cli.lst blocking all grant public
      ```
   b. Bind the utilities.

      If the server is a DRDA host server, bind `ddcsmvs.lst`, `ddcs400.lst`, or `ddcsvm.lst`, depending on the host that you are connecting to (OS/390, AS/400, or VSE&VM). For example:
      ```
      db2 bind @C:\sqllib\bnd\@ddcsmvs.lst blocking all grant public
      ```
      Otherwise, bind the `db2ubind.lst` file:
      ```
      db2 bind @C:\sqllib\bnd\@db2ubind.lst blocking all grant public
      ```
   c. Create the sample table and data for the MTS sample application, as follows:
      ```
      db2 create table account (accountno int, balance int)
      db2 insert into account values(1, 1)
      ```
4. On the DB2 client, ensure that the database manager configuration parameter *tp_mon_name* is set to `MTS`.
5. Run the "BANK" application. Select the **Account** button and the **Visual C++** option, then submit the request. Other options may use SQL that is specific to SQL Server, and may not work.

Appendix F. Using the DB2 Library

The DB2 Universal Database library consists of online help, books (PDF and HTML), and sample programs in HTML format. This section describes the information that is provided, and how you can access it.

To access product information online, you can use the Information Center. For more information, see “Accessing Information with the Information Center” on page 605. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web.

DB2 PDF Files and Printed Books

DB2 Information

The following table divides the DB2 books into four categories:

DB2 Guide and Reference Information
These books contain the common DB2 information for all platforms.

DB2 Installation and Configuration Information
These books are for DB2 on a specific platform. For example, there are separate *Quick Beginnings* books for DB2 on OS/2, Windows, and UNIX-based platforms.

Cross-platform sample programs in HTML
These samples are the HTML version of the sample programs that are installed with the Application Development Client. The samples are for informational purposes and do not replace the actual programs.

Release notes
These files contain late-breaking information that could not be included in the DB2 books.

The installation manuals, release notes, and tutorials are viewable in HTML directly from the product CD-ROM. Most books are available in HTML on the product CD-ROM for viewing and in Adobe Acrobat (PDF) format on the DB2 publications CD-ROM for viewing and printing. You can also order a printed copy from IBM; see “Ordering the Printed Books” on page 601. The following table lists books that can be ordered.

On OS/2 and Windows platforms, you can install the HTML files under the `sqllib\doc\html` directory. DB2 information is translated into different languages; however, all the information is not translated into every language. Whenever information is not available in a specific language, the English information is provided

On UNIX platforms, you can install multiple language versions of the HTML files under the `doc/%L/html` directories, where *%L* represents the locale. For more information, refer to the appropriate *Quick Beginnings* book.

You can obtain DB2 books and access information in a variety of ways:
- "Viewing Information Online" on page 604
- "Searching Information Online" on page 608
- "Ordering the Printed Books" on page 601
- "Printing the PDF Books" on page 600

Table 45. DB2 Information

Name	**Description**	**Form Number** **PDF File Name**	**HTML Directory**
	DB2 Guide and Reference Information		
Administration Guide	*Administration Guide: Planning* provides an overview of database concepts, information about design issues (such as logical and physical database design), and a discussion of high availability. *Administration Guide: Implementation* provides information on implementation issues such as implementing your design, accessing databases, auditing, backup and recovery. *Administration Guide: Performance* provides information on database environment and application performance evaluation and tuning. You can order the three volumes of the *Administration Guide* in the English language in North America using the form number SBOF-8934.	SC09-2946 db2d1x70 SC09-2944 db2d2x70 SC09-2945 db2d3x70	db2d0
Administrative API Reference	Describes the DB2 application programming interfaces (APIs) and data structures that you can use to manage your databases. This book also explains how to call APIs from your applications.	SC09-2947 db2b0x70	db2b0

Table 45. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Application Building Guide	Provides environment setup information and step-by-step instructions about how to compile, link, and run DB2 applications on Windows, OS/2, and UNIX-based platforms.	SC09-2948 db2axx70	db2ax
APPC, CPI-C, and SNA Sense Codes	Provides general information about APPC, CPI-C, and SNA sense codes that you may encounter when using DB2 Universal Database products. Available in HTML format only.	No form number db2apx70	db2ap
Application Development Guide	Explains how to develop applications that access DB2 databases using embedded SQL or Java (JDBC and SQLJ). Discussion topics include writing stored procedures, writing user-defined functions, creating user-defined types, using triggers, and developing applications in partitioned environments or with federated systems.	SC09-2949 db2a0x70	db2a0
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	SC09-2950 db2l0x70	db2l0
Command Reference	Explains how to use the Command Line Processor and describes the DB2 commands that you can use to manage your database.	SC09-2951 db2n0x70	db2n0
Connectivity Supplement	Provides setup and reference information on how to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA application requesters with DB2 Universal Database servers. This book also details how to use DRDA application servers with DB2 Connect application requesters. Available in HTML and PDF only.	No form number db2h1x70	db2h1

Table 45. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Data Movement Utilities Guide and Reference	Explains how to use DB2 utilities, such as import, export, load, AutoLoader, and DPROP, that facilitate the movement of data.	SC09-2955 db2dmx70	db2dm
Data Warehouse Center Administration Guide	Provides information on how to build and maintain a data warehouse using the Data Warehouse Center.	SC26-9993 db2ddx70	db2dd
Data Warehouse Center Application Integration Guide	Provides information to help programmers integrate applications with the Data Warehouse Center and with the Information Catalog Manager.	SC26-9994 db2adx70	db2ad
DB2 Connect User's Guide	Provides concepts, programming, and general usage information for the DB2 Connect products.	SC09-2954 db2c0x70	db2c0
DB2 Query Patroller Administration Guide	Provides an operational overview of the DB2 Query Patroller system, specific operational and administrative information, and task information for the administrative graphical user interface utilities.	SC09-2958 db2dwx70	db2dw
DB2 Query Patroller User's Guide	Describes how to use the tools and functions of the DB2 Query Patroller.	SC09-2960 db2wwx70	db2ww
Glossary	Provides definitions for terms used in DB2 and its components. Available in HTML format and in the *SQL Reference*.	No form number db2t0x70	db2t0
Image, Audio, and Video Extenders Administration and Programming	Provides general information about DB2 extenders, and information on the administration and configuration of the image, audio, and video (IAV) extenders and on programming using the IAV extenders. It includes reference information, diagnostic information (with messages), and samples.	SC26-9929 dmbu7x70	dmbu7
Information Catalog Manager Administration Guide	Provides guidance on managing information catalogs.	SC26-9995 db2dix70	db2di

Table 45. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Information Catalog Manager Programming Guide and Reference	Provides definitions for the architected interfaces for the Information Catalog Manager.	SC26-9997 db2bix70	db2bi
Information Catalog Manager User's Guide	Provides information on using the Information Catalog Manager user interface.	SC26-9996 db2aix70	db2ai
Installation and Configuration Supplement	Guides you through the planning, installation, and setup of platform-specific DB2 clients. This supplement also contains information on binding, setting up client and server communications, DB2 GUI tools, DRDA AS, distributed installation, the configuration of distributed requests, and accessing heterogeneous data sources.	GC09-2957 db2iyx70	db2iy
Message Reference	Lists messages and codes issued by DB2, the Information Catalog Manager, and the Data Warehouse Center, and describes the actions you should take. You can order both volumes of the Message Reference in the English language in North America with the form number SBOF-8932.	Volume 1 GC09-2978 db2m1x70 Volume 2 GC09-2979 db2m2x70	db2m0
OLAP Integration Server Administration Guide	Explains how to use the Administration Manager component of the OLAP Integration Server.	SC27-0787 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	Explains how to create and populate OLAP metaoutlines using the standard OLAP Metaoutline interface (not by using the Metaoutline Assistant).	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	Explains how to create OLAP models using the standard OLAP Model Interface (not by using the Model Assistant).	SC27-0783 db2lpx70	n/a
OLAP Setup and User's Guide	Provides configuration and setup information for the OLAP Starter Kit.	SC27-0702 db2ipx70	db2ip
OLAP Spreadsheet Add-in User's Guide for Excel	Describes how to use the Excel spreadsheet program to analyze OLAP data.	SC27-0786 db2epx70	db2ep

Table 45. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	Describes how to use the Lotus 1-2-3 spreadsheet program to analyze OLAP data.	SC27-0785 db2tpx70	db2tp
Replication Guide and Reference	Provides planning, configuration, administration, and usage information for the IBM Replication tools supplied with DB2.	SC26-9920 db2e0x70	db2e0
Spatial Extender User's Guide and Reference	Provides information about installing, configuring, administering, programming, and troubleshooting the Spatial Extender. Also provides significant descriptions of spatial data concepts and provides reference information (messages and SQL) specific to the Spatial Extender.	SC27-0701 db2sbx70	db2sb
SQL Getting Started	Introduces SQL concepts and provides examples for many constructs and tasks.	SC09-2973 db2y0x70	db2y0
SQL Reference, Volume 1 and Volume 2	Describes SQL syntax, semantics, and the rules of the language. This book also includes information about release-to-release incompatibilities, product limits, and catalog views. You can order both volumes of the *SQL Reference* in the English language in North America with the form number SBOF-8933.	Volume 1 SC09-2974 db2s1x70 Volume 2 SC09-2975 db2s2x70	db2s0
System Monitor Guide and Reference	Describes how to collect different kinds of information about databases and the database manager. This book explains how to use the information to understand database activity, improve performance, and determine the cause of problems.	SC09-2956 db2f0x70	db2f0
Text Extender Administration and Programming	Provides general information about DB2 extenders and information on the administration and configuring of the text extender and on programming using the text extenders. It includes reference information, diagnostic information (with messages) and samples.	SC26-9930 desu9x70	desu9

Table 45. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	GC09-2850 db2p0x70	db2p0
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 7.	SC09-2976 db2q0x70	db2q0
	DB2 Installation and Configuration Information		
DB2 Connect Enterprise Edition for OS/2 and Windows Quick Beginnings	Provides planning, migration, installation, and configuration information for DB2 Connect Enterprise Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2953 db2c6x70	db2c6
DB2 Connect Enterprise Edition for UNIX Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Enterprise Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2952 db2cyx70	db2cy
DB2 Connect Personal Edition Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Personal Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for all supported clients.	GC09-2967 db2c1x70	db2c1
DB2 Connect Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Connect Personal Edition on all supported Linux distributions.	GC09-2962 db2c4x70	db2c4
DB2 Data Links Manager Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for AIX and Windows 32-bit operating systems.	GC09-2966 db2z6x70	db2z6

Table 45. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Enterprise - Extended Edition for UNIX Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2964 db2v3x70	db2v3
DB2 Enterprise - Extended Edition for Windows Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2963 db2v6x70	db2v6
DB2 for OS/2 Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the OS/2 operating system. This book also contains installation and setup information for many supported clients.	GC09-2968 db2i2x70	db2i2
DB2 for UNIX Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2970 db2ixx70	db2ix
DB2 for Windows Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2971 db2i6x70	db2i6
DB2 Personal Edition Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on the OS/2 and Windows 32-bit operating systems.	GC09-2969 db2i1x70	db2i1
DB2 Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on all supported Linux distributions.	GC09-2972 db2i4x70	db2i4

Table 45. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Query Patroller Installation Guide	Provides installation information about DB2 Query Patroller.	GC09-2959 db2iwx70	db2iw
DB2 Warehouse Manager Installation Guide	Provides installation information for warehouse agents, warehouse transformers, and the Information Catalog Manager.	GC26-9998 db2idx70	db2id
	Cross-Platform Sample Programs in HTML		
Sample programs in HTML	Provides the sample programs in HTML format for the programming languages on all platforms supported by DB2. The sample programs are provided for informational purposes only. Not all samples are available in all programming languages. The HTML samples are only available when the DB2 Application Development Client is installed. For more information on the programs, refer to the *Application Building Guide*.	No form number	db2hs
	Release Notes		
DB2 Connect Release Notes	Provides late-breaking information that could not be included in the DB2 Connect books.	See note #2.	db2cr
DB2 Installation Notes	Provides late-breaking installation-specific information that could not be included in the DB2 books.	Available on product CD-ROM only.	
DB2 Release Notes	Provides late-breaking information about all DB2 products and features that could not be included in the DB2 books.	See note #2.	db2ir

Notes:

1. The character *x* in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d0e70` identifies the English version of the *Administration Guide* and the file name `db2d0f70` identifies the French version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

Language	**Identifier**
Brazilian Portuguese	b

Bulgarian	u
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Russian	r
Simp. Chinese	c
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

2. Late breaking information that could not be included in the DB2 books is available in the Release Notes in HTML format and as an ASCII file. The HTML version is available from the Information Center and on the product CD-ROMs. To view the ASCII file:
   - On UNIX-based platforms, see the `Release.Notes` file. This file is located in the `DB2DIR/Readme/%L` directory, where `%L` represents the locale name and `DB2DIR` represents:
     - `/usr/lpp/db2_07_01` on AIX
     - `/opt/IBMdb2/V7.1` on HP-UX, PTX, Solaris, and Silicon Graphics IRIX
     - `/usr/IBMdb2/V7.1` on Linux.
   - On other platforms, see the `RELEASE.TXT` file. This file is located in the directory where the product is installed. On OS/2 platforms, you can also double-click the **IBM DB2** folder and then double-click the **Release Notes** icon.

Printing the PDF Books

If you prefer to have printed copies of the books, you can print the PDF files found on the DB2 publications CD-ROM. Using the Adobe Acrobat Reader, you can print either the entire book or a specific range of pages. For the file name of each book in the library, see Table 45 on page 592.

You can obtain the latest version of the Adobe Acrobat Reader from the Adobe Web site at http://www.adobe.com.

The PDF files are included on the DB2 publications CD-ROM with a file extension of PDF. To access the PDF files:

1. Insert the DB2 publications CD-ROM. On UNIX-based platforms, mount the DB2 publications CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
2. Start the Acrobat Reader.
3. Open the desired PDF file from one of the following locations:
   - On OS/2 and Windows platforms:

     `x:\doc\language` directory, where *x* represents the CD-ROM drive and *language* represent the two-character country code that represents your language (for example, EN for English).
   - On UNIX-based platforms:

     `/cdrom/doc/%L` directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

You can also copy the PDF files from the CD-ROM to a local or network drive and read them from there.

Ordering the Printed Books

You can order the printed DB2 books either individually or as a set (in North America only) by using a sold bill of forms (SBOF) number. To order books, contact your IBM authorized dealer or marketing representative, or phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada. You can also order the books from the Publications Web page at http://www.elink.ibmlink.ibm.com/pbl/pbl.

Two sets of books are available. SBOF-8935 provides reference and usage information for the DB2 Warehouse Manager. SBOF-8931 provides reference and usage information for all other DB2 Universal Database products and features. The contents of each SBOF are listed in the following table:

Table 46. Ordering the printed books

SBOF Number	Books Included	
SBOF-8931	• Administration Guide: Planning • Administration Guide: Implementation • Administration Guide: Performance • Administrative API Reference • Application Building Guide • Application Development Guide • CLI Guide and Reference • Command Reference • Data Movement Utilities Guide and Reference • Data Warehouse Center Administration Guide • Data Warehouse Center Application Integration Guide • DB2 Connect User's Guide • Installation and Configuration Supplement • Image, Audio, and Video Extenders Administration and Programming • Message Reference, Volumes 1 and 2	• OLAP Integration Server Administration Guide • OLAP Integration Server Metaoutline User's Guide • OLAP Integration Server Model User's Guide • OLAP Integration Server User's Guide • OLAP Setup and User's Guide • OLAP Spreadsheet Add-in User's Guide for Excel • OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3 • Replication Guide and Reference • Spatial Extender Administration and Programming Guide • SQL Getting Started • SQL Reference, Volumes 1 and 2 • System Monitor Guide and Reference • Text Extender Administration and Programming • Troubleshooting Guide • What's New
SBOF-8935	• Information Catalog Manager Administration Guide • Information Catalog Manager User's Guide • Information Catalog Manager Programming Guide and Reference	• Query Patroller Administration Guide • Query Patroller User's Guide

DB2 Online Documentation

Accessing Online Help

Online help is available with all DB2 components. The following table describes the various types of help.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: `? command` where *command* represents a keyword or the entire command. For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.
Client Configuration Assistant Help *Command Center Help* *Control Center Help* *Data Warehouse Center Help* *Event Analyzer Help* *Information Catalog Manager Help* *Satellite Administration Center Help* *Script Center Help*	Explains the tasks you can perform in a window or notebook. The help includes overview and prerequisite information you need to know, and it describes how to use the window or notebook controls.	From a window or notebook, click the **Help** push button or press the **F1** key.
Message Help	Describes the cause of a message and any action you should take.	From the command line processor in interactive mode, enter: `? XXXnnnnn` where *XXXnnnnn* represents a valid message identifier. For example, `? SQL30081` displays help about the SQL30081 message. To view message help one screen at a time, enter: `? XXXnnnnn | more` To save message help in a file, enter: `? XXXnnnnn > filename.ext` where *filename.ext* represents the file where you want to save the message help.

Type of Help	Contents	How to Access...
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: `help` *`statement`* where *statement* represents an SQL statement. For example, `help SELECT` displays help about the SELECT statement. **Note:** SQL help is not available on UNIX-based platforms.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: `?` *`sqlstate`* `or ?` *`class code`* where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state. For example, `? 08003` displays help for the 08003 SQL state, while `? 08` displays help for the 08 class code.

Viewing Information Online

The books included with this product are in Hypertext Markup Language (HTML) softcopy format. Softcopy format enables you to search or browse the information and provides hypertext links to related information. It also makes it easier to share the library across your site.

You can view the online books or sample programs with any browser that conforms to HTML Version 3.2 specifications.

To view online books or sample programs:
- If you are running DB2 administration tools, use the Information Center.
- From a browser, click **File —>Open Page**. The page you open contains descriptions of and links to DB2 information:
  - On UNIX-based platforms, open the following page:
    ```
    INSTHOME/sqllib/doc/%L/html/index.htm
    ```
    where `%L` represents the locale name.
  - On other platforms, open the following page:
    ```
    sqllib\doc\html\index.htm
    ```
    The path is located on the drive where DB2 is installed.

If you have not installed the Information Center, you can open the page by double-clicking the **DB2 Information** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

Installing the Netscape Browser

If you do not already have a Web browser installed, you can install Netscape from the Netscape CD-ROM found in the product boxes. For detailed instructions on how to install it, perform the following:

1. Insert the Netscape CD-ROM.
2. On UNIX-based platforms only, mount the CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
3. For installation instructions, refer to the `CDNAV`*nn*`.txt` file, where *nn* represents your two character language identifier. The file is located at the root directory of the CD-ROM.

Accessing Information with the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on all platforms on which the DB2 administration tools are available.

You can open the Information Center by double-clicking the Information Center icon. Depending on the system you are using, the icon is in the Information folder in the main product folder or the Windows **Start** menu.

You can also access the Information Center by using the toolbar and the **Help** menu on the DB2 Windows platform.

The Information Center provides six types of information. Click the appropriate tab to look at the topics provided for that type.

Tasks	Key tasks you can perform using DB2.
Reference	DB2 reference information, such as keywords, commands, and APIs.
Books	DB2 books.
Troubleshooting	Categories of error messages and their recovery actions.
Sample Programs	Sample programs that come with the DB2 Application Development Client. If you did not install the DB2 Application Development Client, this tab is not displayed.
Web	DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides a find feature, so you can look for a specific topic without browsing the lists.

For a full text search, follow the hypertext link in the Information Center to the **Search DB2 Online Information** search form.

The HTML search server is usually started automatically. If a search in the HTML information does not work, you may have to start the search server using one of the following methods:

On Windows
: Click **Start** and select **Programs —> IBM DB2 —> Information —> Start HTML Search Server**.

On OS/2
: Double-click the **DB2 for OS/2** folder, and then double-click the **Start HTML Search Server** icon.

Refer to the release notes if you experience any other problems when searching the HTML information.

Note: The Search function is not available in the Linux, PTX, and Silicon Graphics IRIX environments.

Using DB2 Wizards

Wizards help you complete specific administration tasks by taking you through each task one step at a time. Wizards are available through the Control Center and the Client Configuration Assistant. The following table lists the wizards and describes their purpose.

Note: The Create Database, Create Index, Configure Multisite Update, and Performance Configuration wizards are available for the partitioned database environment.

Wizard	Helps You to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click **Add**.
Backup Database	Determine, create, and schedule a backup plan.	From the Control Center, right-click the database you want to back up and select **Backup —> Database Using Wizard**.

Wizard	Helps You to...	How to Access...
Configure Multisite Update	Configure a multisite update, a distributed transaction, or a two-phase commit.	From the Control Center, right-click the **Databases** folder and select **Multisite Update**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, right-click the **Databases** folder and select **Create —> Database Using Wizard**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, right-click the **Tables** icon and select **Create —> Table Using Wizard**.
Create Table Space	Create a new table space.	From the Control Center, right-click the **Table Spaces** icon and select **Create —> Table Space Using Wizard**.
Create Index	Advise which indexes to create and drop for all your queries.	From the Control Center, right-click the **Index** icon and select **Create —> Index Using Wizard**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, right-click the database you want to tune and select **Configure Performance Using Wizard**. For the partitioned database environment, from the Database Partitions view, right-click the first database partition you want to tune and select **Configure Performance Using Wizard**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, right-click the database you want to restore and select **Restore —> Database Using Wizard**.

Setting Up a Document Server

By default, the DB2 information is installed on your local system. This means that each person who needs access to the DB2 information must install the same files. To have the DB2 information stored in a single location, perform the following steps:

1. Copy all files and subdirectories from `\sqllib\doc\html` on your local system to a Web server. Each book has its own subdirectory that contains all the necessary HTML and GIF files that make up the book. Ensure that the directory structure remains the same.

2. Configure the Web server to look for the files in the new location. For information, refer to the NetQuestion Appendix in the *Installation and Configuration Supplement*.
3. If you are using the Java version of the Information Center, you can specify a base URL for all HTML files. You should use the URL for the list of books.
4. When you are able to view the book files, you can bookmark commonly viewed topics. You will probably want to bookmark the following pages:
   - List of books
   - Tables of contents of frequently used books
   - Frequently referenced articles, such as the ALTER TABLE topic
   - The Search form

For information about how you can serve the DB2 Universal Database online documentation files from a central machine, refer to the NetQuestion Appendix in the *Installation and Configuration Supplement*.

Searching Information Online

To find information in the HTML files, use one of the following methods:

- Click **Search** in the top frame. Use the search form to find a specific topic. This function is not available in the Linux, PTX, or Silicon Graphics IRIX environments.
- Click **Index** in the top frame. Use the index to find a specific topic in the book.
- Display the table of contents or index of the help or the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. See "Accessing Information with the Information Center" on page 605 for details.

Appendix G. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

Trademarks

The following terms, which may be denoted by an asterisk(*), are trademarks of International Business Machines Corporation in the United States, other countries, or both.

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, other countries, or both.

Index

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. This guide suggests information that you can gather to help DB2 Customer Support to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

If you live in the U.S.A., then you can call one of the following numbers:
- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

ftp.software.ibm.com
: Log on as anonymous. In the directory `/ps/products/db2`, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l
: These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2
: Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the *IBM Software Support Handbook*. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-2945-00