IBM® DB2 Universal Database™

Administration Guide: Performance

Version 8

SC09-4821-00

IBM® DB2 Universal Database™

Administration Guide: Performance

Version 8

SC09-4821-00

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

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this book ix
Who should use this book. x
How this book is structured x
A brief overview of the other Administration Guide volumes xi
Administration Guide: Planning xi
Administration Guide: Implementation . . xii

Part 1. Introduction to performance 1

Chapter 1. Introduction to performance . . 3
Elements of performance 3
Performance-tuning guidelines 4
The performance-tuning process 5
Developing a performance-improvement process 5
Performance information that users can provide 6
Performance-tuning limits. 7
Quick-start tips for performance tuning . . . 7

Chapter 2. Architecture and processes . . 11
DB2 architecture and process overview . . . 11
Deadlocks between applications 14
Disk storage overview 15
Disk-storage performance factors 15
Database directories and files 16
Table space overview 19
Table spaces 19
SMS table spaces 19
DMS table spaces 20
Illustration of the DMS table-space address map. 22
Tables and indexes. 23
Table and index management for standard tables 23
Table and index management for MDC tables 28
Index structure 31
Processes 32
Logging process. 33
Insert process 34
Update process 35
Client-server processing model. 36
Memory management. 44

Part 2. Tuning application performance 49

Chapter 3. Application considerations . . 51
Concurrency control and isolation levels . . 51
Concurrency issues. 51
Performance impact of isolation levels . . 52
Specifying the isolation level 57
Concurrency control and locking 59
Locks and concurrency control. 59
Lock attributes 61
Locks and performance 63
Guidelines for locking. 68
Correcting lock escalation problems . . . 70
Lock type compatibility 72
Lock modes and access paths for standard tables 74
Lock modes for table and RID index scans of MDC tables 77
Locking for block index scans for MDC tables 80
Factors that affect locking 83
Factors that affect locking 84
Locks and types of application processing 84
Locks and data-access methods 85
Index types and next-key locking 86
Optimization factors 88
Optimization class guidelines 88
Optimization classes 90
Setting the optimization class 93
Tuning applications 95
Guidelines for restricting select statements 95
Specifying row blocking to reduce overhead 99
Query tuning guidelines 101
Efficient SELECT statements 101
Compound SQL guidelines 103
Character-conversion guidelines 105
Guidelines for stored procedures. . . . 106
Parallel processing for applications . . . 107

Chapter 4. Environmental considerations 111

Database partition group impact on query optimization 111
Table space impact on query optimization 111
Server options affecting federated databases 115

Chapter 5. System catalog statistics. . . 117
Catalog statistics 117
Collecting and analyzing catalog statistics 119
Guidelines for collecting and updating statistics 119
Collecting catalog statistics. 120
Collecting distribution statistics for specific columns 121
Collecting index statistics 123
Statistics collected. 124
Catalog statistics tables 124
Statistical information that is collected 131
Distribution statistics. 134
Optimizer use of distribution statistics 136
Extended examples of distribution-statistics use 141
Detailed index statistics. 142
Sub-element statistics 144
Catalog statistics that users can update . . 145
Statistics for user-defined functions . . . 146
Catalog statistics for modeling and what-if planning 147
Statistics for modeling production databases 149
General rules for updating catalog statistics manually 152
Rules for updating column statistics manually 153
Rules for updating distribution statistics manually 153
Rules for updating table and nickname statistics manually 154
Rules for updating index statistics manually 155

Chapter 6. Understanding the SQL compiler 159
The SQL compiler process 159
Configuration parameters that affect query optimization 163
Query rewriting 166
Query rewriting methods and examples 166
Compiler rewrite example: view merges 168
Compiler rewrite example: DISTINCT elimination 171
Compiler rewrite example: implied predicates 173
Column correlation for multiple predicates 174
Data access methods. 176
Data-access methods and concepts . . . 176
Data access through index scans 177
Types of index access 181
Index access and cluster ratios 183
Predicate terminology 184
Join methods and strategies 186
Joins 187
Join methods 190
Strategies for selecting optimal joins . . 191
Replicated materialized-query tables in partitioned databases 194
Join strategies in partitioned databases 196
Join methods in partitioned databases . . 198
Effects of sorting and grouping 204
Optimization strategies 206
Optimization strategies for intra-partition parallelism 206
Optimization strategies for MDC tables 209
Automatic summary tables. 210
Federated database query-compiler phases 213
Federated database pushdown analysis 213
Guidelines for analyzing where a federated query is evaluated 218
Remote SQL generation and global optimization in federated databases. . . 220
Global analysis of federated database queries 223

Chapter 7. SQL Explain facility 227
SQL explain facility 227
Tools for collecting and analyzing explain information 228
Explain tools 228
Guidelines for using explain information 230
Explain information collected 232
The explain tables and organization of explain information 232
Explain information for data objects . . 234
Explain information for data operators 234
Explain information for instances . . . 235
Guidelines for capturing explain information 239
Guidelines for analyzing explain information 241
The Design Advisor 242

Part 3. Tuning and configuring your system 249

Chapter 8. Operational performance . . . 251
Memory usage 251
Organization of memory use 251
Database manager shared memory . . . 254
The FCM buffer pool and memory requirements 256
Global memory and parameters that control it 258
Guidelines for tuning parameters that affect memory usage 261
Buffer pools 263
Buffer-pool management 264
Secondary buffer pools in extended memory on 32-bit platforms 266
Buffer-pool management of data pages 267
Illustration of buffer-pool data-page management 269
Management of multiple database buffer pools 271
Prefetching concepts 274
Prefetching data into the buffer pool . . 274
Sequential prefetching 275
Block-based buffer pools for improved sequential prefetching 277
List prefetching 278
I/O management 279
I/O server configuration for prefetching and parallelism 279
Illustration of prefetching with parallel I/O 280
Parallel I/O management 282
Guidelines for sort performance 284
Table management 287
Table reorganization 287
Determining when to reorganize tables 288
Choosing a table reorganization method 291
Index management 294
Advantages and disadvantages of indexes 294
Index planning tips 296
Index performance tips 299
Index cleanup and maintenance 302
Index reorganization 303
Online index defragmentation 305
DMS device considerations 307
Agent management 308
Database agents 308
Database-agent management 310
Configuration parameters that affect the number of agents 311
Connection-concentrator improvements for client connections 312
Agents in a partitioned database 315
The database system-monitor information 316

Chapter 9. Using the governor 319
The Governor utility 319
Governor startup and shutdown 320
Starting and stopping the governor . . . 320
The Governor daemon 321
Governor configuration 322
Configuring the Governor 323
The Governor configuration file 324
Governor rule elements 326
Example of a Governor configuration file 331
Governor log-file use 333
Governor log files 333
Governor log-file queries 334

Chapter 10. Scaling your configuration 337
Management of database server capacity . . 337
Partitions in a partitioned database 338
Adding a partition to a running database system 339
Adding a partition to a stopped database system on Windows NT 341
Adding a partition to a stopped database system on UNIX 342
Node-addition error recovery 344
Dropping a database partition 346

Chapter 11. Redistributing Data Across Database Partitions 347
Data redistribution 347
Determining whether to redistribute data 349
Redistributing data across partitions . . . 350
Log space requirements for data redistribution 352
Redistribution-error recovery 353

Chapter 12. Benchmark testing 355
Benchmark testing 355
Benchmark preparation 356
Benchmark test creation 358
Examples of db2batch tests 360
Benchmark test execution 364
Benchmark test analysis example 366

Chapter 13. Configuring DB2 369
Configuration parameters 369
Configuration parameter tuning 371
Configuring DB2 with configuration parameters 372
Configuration parameters summary. 376
Database Manager Configuration Parameter Summary 376
Database Configuration Parameter Summary 382
DB2 Administration Server (DAS) Configuration Parameter Summary . . . 388
Parameter Details by Function 390
Capacity Management 391
Database Shared Memory 391
Application Shared Memory 401
Agent Private Memory 405
Agent/Application Communication Memory 416
Database Manager Instance Memory . . 421
Locks 427
I/O and Storage 431
Agents 438
Stored Procedures and User Defined Functions 450
Logging and Recovery 454
Database Log Files 454
Database Log Activity 464
Recovery. 469
Distributed Unit of Work Recovery . . . 476
Database Management 480
Query Enabler 480
Attributes 481
DB2 Data Links Manager 483
Status. 486
Compiler Settings. 489
Communications 496
Communication Protocol Setup 496
DB2 Discovery. 498
Partitioned Database Environment 501
Communications 501
Parallel Processing 505
Instance Management 507
Diagnostic 507
Database System Monitor Parameters . . 511
System Management. 512
Instance Administration. 520
DB2 Administration Server. 529
DAS Discovery Mode configuration parameter - discover. 530
Name of the DB2 Server System configuration parameter - db2system . . 530
DAS Administration Authority Group Name configuration parameter - dasadm_group. 531
Scheduler Mode configuration parameter - sched_enable 532
Tools Catalog Database Instance configuration parameter - toolscat_inst. . 532
Tools Catalog Database configuration parameter - toolscat_db. 533
Tools Catalog Database Schema configuration parameter - toolscat_schema 533
SMTP Server configuration parameter - smtp_server. 534
Java Development Kit Installation Path DAS configuration parameter - jdk_path . 535
Execute Expired Tasks configuration parameter - exec_exp_task 535
Scheduler User ID configuration parameter - sched_userid 536
Location of Contact List configuration parameter - contact_host 536
Authentication Type DAS configuration parameter - authentication. 537
DAS Code Page configuration parameter - das_codepage 537
DAS Territory configuration parameter - das_territory 538

Part 4. Appendixes 539

Appendix A. DB2 Registry and Environment Variables 541
DB2 registry and environment variables . . 541
Registry and environment variables by category 542
General registry variables 542
System environment variables 546
Communications variables. 548
Command-line variables 553
MPP configuration variables 554
SQL compiler variables 556
Performance variables 562
Data-links variables 569
Miscellaneous variables. 571

Appendix B. Explain tables 577
Explain tables 577

EXPLAIN_ARGUMENT table 578
EXPLAIN_INSTANCE table 582
EXPLAIN_OBJECT table 585
EXPLAIN_OPERATOR table 588
EXPLAIN_PREDICATE table 590
EXPLAIN_STATEMENT table. 592
EXPLAIN_STREAM table 595
ADVISE_INDEX table 597
ADVISE_WORKLOAD table 600

Appendix C. SQL explain tools 601
SQL explain tools 601
db2expln. 602
db2expln syntax and parameters. . . . 602
Usage notes for db2expln 608
dynexpln 610
Explain output information 610
Description of db2expln and dynexpln output 610
Table access information 611
Temporary table information 617
Join information 620
Data stream information 622
Insert, update, and delete information 623
Block and row identifier preparation information 624
Aggregation information 625
Parallel processing information 626
Federated query information 629
Miscellaneous information 630
Examples of db2expln and dynexpln Output 633
Examples of db2expln and dynexpln output 633
Example one: no parallelism 633
Example two: single-partition plan with intra-partition parallelism 635
Example three: multipartition plan with inter-partition parallelism 637
Example four: multipartition plan with inter-partition and intra-partition parallelism 640
Example five: federated database plan 642

Appendix D. db2exfmt - Explain table-format tool. 645

Appendix E. DB2 Universal Database technical information 647
Overview of DB2 Universal Database technical information 647
Categories of DB2 technical information 648
Printing DB2 books from PDF files 655
Ordering printed DB2 books 656
Accessing online help 656
Finding topics by accessing the DB2 Information Center from a browser 658
Finding product information by accessing the DB2 Information Center from the administration tools 660
Viewing technical documentation online directly from the DB2 HTML Documentation CD. 661
Updating the HTML documentation installed on your machine 662
Copying files from the DB2 HTML Documentation CD to a Web Server. . . . 664
Troubleshooting DB2 documentation search with Netscape 4.x. 664
Searching the DB2 documentation 665
Online DB2 troubleshooting information . . 666
Accessibility 667
Keyboard Input and Navigation 667
Accessible Display 668
Alternative Alert Cues 668
Compatibility with Assistive Technologies 668
Accessible Documentation 668
DB2 tutorials 668
DB2 Information Center for topics 669

Appendix F. Notices 671
Trademarks 674

Index 677

Contacting IBM 691
Product information 691

About this book

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

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

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

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

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

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

- The Alert Center to monitor the messages that result from other DB2 operations.
- The Health Center provides a tool to assist DBAs in the resolution of performance and resource allocation problems.
- The Tools Settings to change the settings for the Control Center, Alert Center, and Replication.
- The Journal to schedule jobs that are to run unattended.
- The Data Warehouse Center to manage warehouse objects.

Who should use this book

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

How this book is structured

This book contains information about the following major topics:

Introduction to Performance

- Chapter 1, “Introduction to performance”, introduces concepts and considerations for managing and improving DB2 UDB performance.
- Chapter 2, “Architecture and processes”, 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”, 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 database manager, database, and DAS configuration parameters.

Appendixes

- Appendix A, “DB2 Registry and Environment Variables”, presents profile registry values and environment variables.
- Appendix B, “Explain tables”, The explain table section 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.

A brief overview of the other Administration Guide volumes

Administration Guide: Planning

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

Database Concepts

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

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.

Appendixes

- ″Incompatibilities Between Releases″ presents the incompatibilities introduced by Version 7 and Version 8, as well as future incompatibilities that you should be aware of.
- ″National Language Support (NLS)″ describes DB2 National Language Support, including information about territories, 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:

Implementing Your Design

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

Database Security

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

Appendixes

- ″Naming Rules″ presents the rules to follow when naming databases and objects.

- ″Lightweight Directory Access Protocol (LDAP) Directory Services″ provides information about how you can use LDAP Directory Services.
- ″Issuing Commands to Multiple Database Partition″ discusses the use of the *db2_all* and *rah* shell scripts to send commands to all partitions in a partitioned database environment.
- ″Windows Management Instrumentation (WMI) Support″ describes how DB2 supports this management infrastructure standard to integrate various hardware and software management systems. Also discussed is how DB2 integrates with WMI.
- ″How DB2 for Windows NT Works with Windows NT Security″ describes how DB2 works with Windows NT security.
- ″Using the Windows Performance Monitor″ provides information about registering DB2 with the Windows NT Performance Monitor, and using the performance information.
- ″Working with Windows Database Partition Servers″ provides information about the utilities available to work with database partition servers on Windows NT or Windows 2000.
- ″Configuring Multiple Logical Nodes″ describes how to configure multiple logical nodes in a partitioned database environment.
- ″Extending the Control Center″ provides information about how you can extend the Control Center by adding new tool bar buttons including new actions, adding new object definitions, and adding new action definitions.

Note: Two chapters have been removed from this book.

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

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

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

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

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

Part 1. Introduction to performance

Chapter 1. Introduction to performance

The sections in this chapter describe performance tuning and provide some suggestions for:

- Creating a performance monitoring and tuning plan
- Using user information about performance problems
- Getting a quick start on initial performance tuning

Elements of performance

Performance is the way a computer system behaves given a particular work load. Performance is measured in terms of system response time, throughput, and availability. Performance is also affected by:

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

In general, you tune your system to improve its cost-benefit ratio. Specific goals could include:

- Processing a larger, or more demanding, work load without increasing processing costs

  For example, to increase 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 degrading service to your users

Translating performance from technical terms to economic terms is difficult. Performance tuning certainly costs money in terms of user time as well as 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.

Related concepts:

- “Performance-tuning guidelines” on page 4
- “Quick-start tips for performance tuning” on page 7

Related tasks:

- "Developing a performance-improvement process" on page 5

Performance-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. By changing only one at a time, this allows you to have a benchmark to evaluate whether the change does what you want.

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 Manager
- SQL Statements
- Application Programs

Check for hardware as well as 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 fall-back 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.

Related concepts:

- "Elements of performance" on page 3
- "Quick-start tips for performance tuning" on page 7

Related tasks:

- "Developing a performance-improvement process" on page 5

The performance-tuning process

You develop a performance monitoring and tuning plan, taking user input into account, and recognizing the limits of tuning in your system.

Developing a performance-improvement process

The performance-improvement process is an iterative, long-term approach to monitoring and tuning aspects of performance. Depending on the result of monitoring, you and your performance team adjust the configuration of the database server and make changes to the applications that use the database server.

Base your performance monitoring and tuning decisions on your knowledge of the kinds of applications that use the data and the patterns of data access. Different kinds of applications have different performance requirements.

Consider the following outline of the performance-improvement process as a guideline.

Procedure:

To develop a performance-improvement process:

1. Define performance objectives.
2. Establish performance indicators for the major constraints in the system.
3. Develop and execute a performance monitoring plan.
4. Continually analyze the results of monitoring to determine which resources require tuning.
5. Make one adjustment at a time.

   Even if you think that more than one resource requires tuning, or if several tuning options are available for the resource you want to tune, make only one change at a time so that you can make sure that your tuning efforts are producing the effect you want. At some point, you can no longer improve performance by tuning the database server and applications. Then you need to upgrade your hardware.

Actual performance tuning requires trade-offs among system resources. For example, to provide improved I/O performance you might increase buffer pool sizes, but larger buffer pools require more memory, which might degrade other aspects of performance.

Related concepts:

- "Elements of performance" on page 3
- "Performance-tuning guidelines" on page 4
- "Quick-start tips for performance tuning" on page 7
- "Performance-tuning limits" on page 7
- "Performance information that users can provide" on page 6

Performance information that users can provide

The first sign that your system requires tuning might be complaints from users. 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. You can usually determine where to start looking for a problem by asking a few simple questions. For example, you might ask your users:

- What do you mean by "slow response"? Is it 10 % slower than you expect it to be, or tens of times slower?
- When did you notice the problem? Is it recent or has it always been there?
- Do other users have the same problem? Are these users one or two individuals or a whole group?
- If a group of users is experiencing the same problems, are they connected to the same local area network?

- Do the the problems seem to be related to a specific transaction or application program?
- Do you notice any pattern in the problem occurrence? For example, does the problem occur at a specific time of day, such as during lunch hour, or is it more or less continuous?

Related concepts:

- "Performance-tuning guidelines" on page 4

Related tasks:

- "Developing a performance-improvement process" on page 5

Performance-tuning limits

Tuning can make only a certain amount of change in the efficiency of a system. Consider how much time and money you should spend on improving system performance, and how much spending additional time and money will help the users of the system.

For example, tuning can often improve performance if the system encounters a performance bottleneck. If you are close to the performance limits of your system and the number of users increases by about ten percent, the response time is likely to increase by much more than ten percent. In this situation, you need to determine how to counterbalance this degradation in performance by tuning your system.

However, there is a point beyond which tuning cannot help. At this point, consider revising your goals and expectations within the limits of your environment. For significant performance improvements, you might need to add more disk storage, faster CPU, additional CPUs, more main memory, faster communication links, or a combination of these.

Related concepts:

- "Management of database server capacity" on page 337

Related tasks:

- "Developing a performance-improvement process" on page 5

Quick-start tips for performance tuning

When you start a new instance of DB2, consider the following suggestions for a basic configuration:

- Use the Configuration Advisor in the Control Center to get advice about reasonable beginning defaults for your system. The defaults shipped with DB2® should be tuned for your unique hardware environment.

  Gather information about the hardware at your site so you can answer the wizard questions. You can apply the suggested configuration parameter settings immediately or let the wizard create a script based on your answers and run the script later.

  This script also provides a list of the most commonly tuned parameters for later reference.
- Use other wizards in the Control Center and Client Configuration Assistant for performance-related administration tasks. These tasks are usually those in which you can achieve significant performance improvements by spending spend a little time and effort.

  Other wizards can help you improve performance of individual tables and general data access. These wizards include the Create Database, Create Table, Index, and Configure Multisite Update wizards. The Workload Performance wizard guides you through tasks that help tune queries. The Health Center provides a set of monitoring and tuning tools.
- Use the Design Advisor tool (`db2advis`) to find out what indexes will improve query performance.
- Use the ACTIVATE DATABASE command to start databases. In a partitioned database, this command activates the database on all partitions and avoids the startup time required to initialize the database when the first application connects.

  Note: If you use the ACTIVATE DATABASE command, you must shut down the database with the DEACTIVATE DATABASE command. The last application that disconnects from the database does not shut it down.
- Consult the summary tables that list and briefly describe each configuration parameter available for the database manager and each database.

  These summary tables contain a column that indicates whether tuning the parameter results in high, medium, low, or no performance changes, either for better or for worse. Use this table to find the parameters that you might tune for the largest performance improvements.

Related concepts:

- "The database system-monitor information" on page 316

Related tasks:

- "Creating a database with default or tailored settings using a wizard: Control Center help" in the *Help: Control Center*

- “Configuring database parameters for performance using a wizard: Control Center help” in the *Help: Control Center*
- “Improving performance for a database with different workloads using a wizard: Control Center help” in the *Help: Control Center*

Related reference:

- “Configuration parameters summary” on page 376

Chapter 2. Architecture and processes

This chapter provides general information about the DB2 architecture and process schema.

DB2 architecture and process overview

General information about DB2® architecture and processes can help you understand detailed information provided for specific topics.

The following figure shows a general overview of the architecture and processes for DB2 UDB.

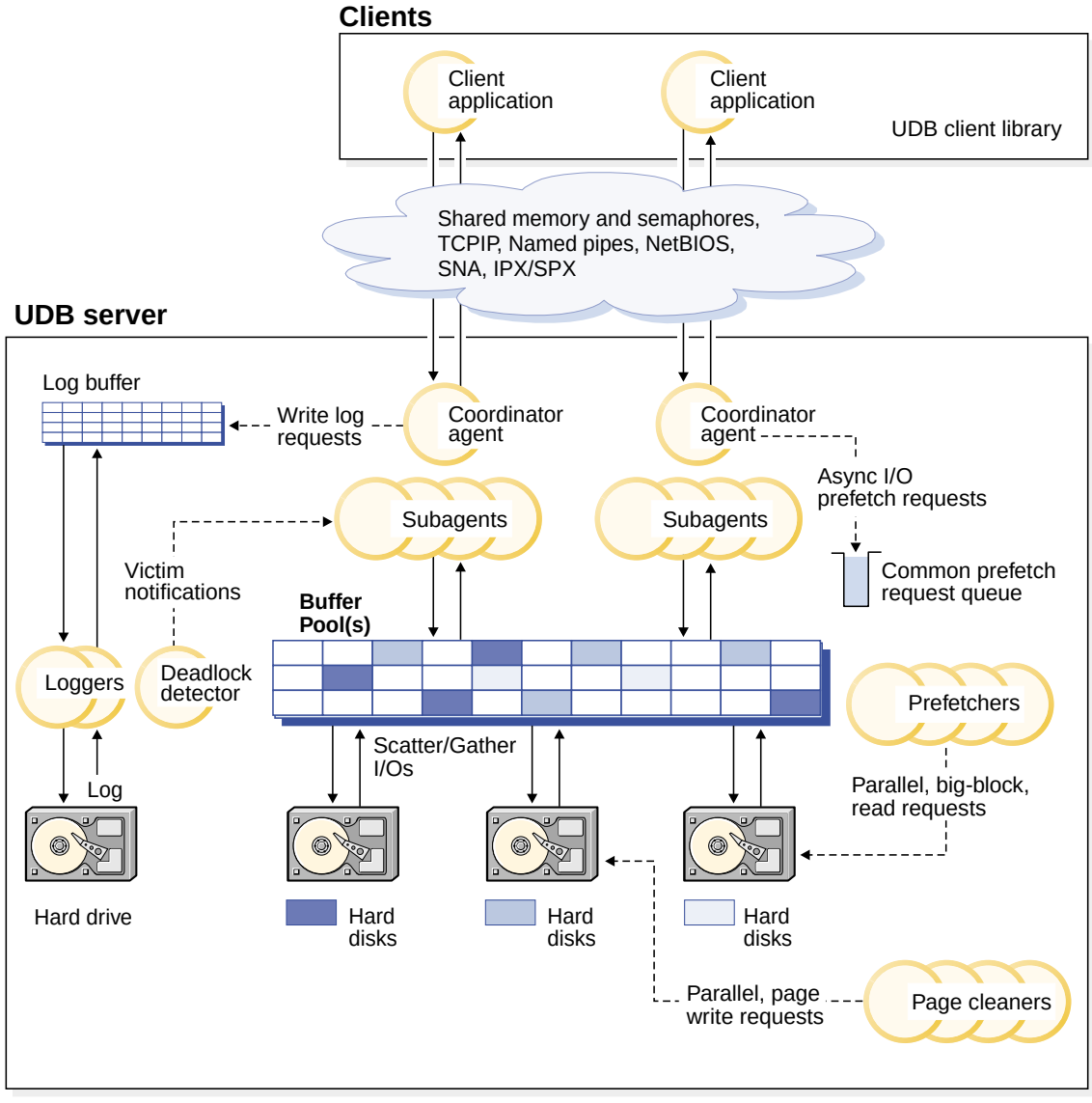


Figure 1. Architecture and Processes Overview

On the client side, either local or remote applications, or both, are linked with the DB2 Universal Database™ client library. Local clients communicate using shared memory and semaphores; remote clients use a protocol such as Named Pipes (NPIPE), TCP/IP, NetBIOS, or SNA.

On the server side, activity is controlled by engine dispatchable units (EDUs). In all figures in this section, EDUs are shown as circles or groups of circles.

EDUs are implemented as threads in a single process on Windows-based platforms and as processes on UNIX. DB2 agents are the most common type of EDUs. These agents perform most of the SQL processing on behalf of applications. Prefetchers and page cleaners are other common EDUs.

A set of subagents might be assigned to process the client application requests. Multiple subagents can be assigned if the machine where the server resides has multiple processors or is part of a partitioned database. For example, in a symmetric multiprocessing (SMP) environment, multiple SMP subagents can exploit the many processors.

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

Buffer pools are areas of database server memory where database pages of user table data, index data, and catalog data are temporarily moved and can be modified. Buffer pools are a key determinant of database performance because data can be accessed much faster from memory than from disk. If more of the data needed by applications is present in a buffer pool, less time is required to access the data than to find it on disk.

The configuration of the buffer pools, as well as prefetcher and page cleaner EDUs, controls how quickly data can be accessed and how readily available it is to applications.

- **Prefetchers** 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. If you have multiple disks for storage of the database data, the data can be striped across the disks. Striping data lets the prefetchers use multiple disks at the same time to retrieve data.
- **Page cleaners** move data from the buffer pool back out to disk. Page cleaners are background EDUs that are independent of the application agents. They look for pages from the buffer pool that are no longer needed and write the pages to disk. Page cleaners ensure that there is room in the buffer pool for the pages being retrieved by the prefetchers.

Without the independent prefetchers and the 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.

Related concepts:

- “Prefetching data into the buffer pool” on page 274
- “Deadlocks between applications” on page 14
- “Database directories and files” on page 16
- “Logging process” on page 33
- “Update process” on page 35
- “Client-server processing model” on page 36
- “Memory management” on page 44
- “Connection-concentrator improvements for client connections” on page 312

Related reference:

- “Maximum Number of Coordinating Agents configuration parameter - max_coordagents” on page 446
- “Maximum Number of Client Connections configuration parameter - max_connections” on page 448

Deadlocks between applications

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 created when one application is waiting for another application to release a lock on data. Each of the waiting applications is locking data needed by another application. Mutual waiting for the other application to release a lock on held data leads to a deadlock. The applications can wait forever until one application releases the lock on the held data.

A deadlock is illustrated in the following figure.

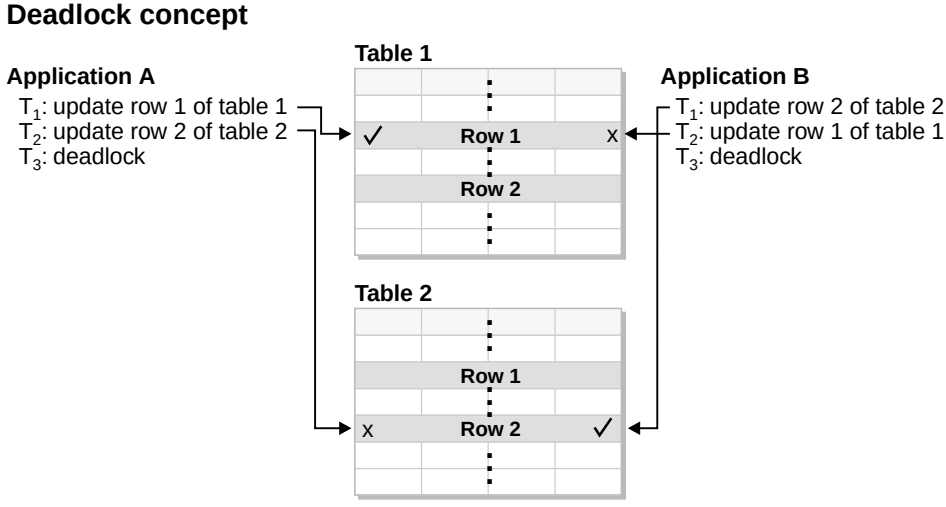


Figure 2. Deadlock detector

Because applications do not voluntarily release locks on data that they need, a deadlock detector process is required to break deadlocks and allow application processing to continue. 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 and releases the locks currently held by that "volunteered" application. By releasing the locks of that application, the data required by other waiting applications is made available for use. The waiting applications can then access the data required to complete transactions.

Related concepts:

- "Locks and performance" on page 63
- "DB2 architecture and process overview" on page 11

Disk storage overview

Understanding how data is stored on disk helps you tune I/O.

Disk-storage performance factors

The hardware that makes up your system can influence the performance of your system. As an example of the influence of hardware on performance, consider some of the implications associated with disk storage.

Four aspects of disk-storage management affect performance:

- **Division of storage**

How you divide a limited amount of storage between indexes and data and among table spaces 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 wasted storage is a resource that could be used to improve performance elsewhere.

- **Distribution of 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.

- **Lack of available storage**

  Reaching the limit of available storage can degrade overall performance.

Related concepts:

- "DMS device considerations" on page 307
- "Database directories and files" on page 16
- "SMS table spaces" on page 19
- "DMS table spaces" on page 20
- "Table and index management for standard tables" on page 23
- "Table and index management for MDC tables" on page 28

Database directories and files

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

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

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

and subsequent databases are given higher numbers: SQL00002, and so on. These subdirectories differentiate databases created in this instance on the directory that you specified in the CREATE DATABASE command.

The directory structure appears as follows:

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

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

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

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

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

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

Additional information for SMS database directories

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

- SQLT0000.0 subdirectory contains the catalog table space with the system catalog tables.
- SQLT0001.0 subdirectory contains the default temporary table space.
- SQLT0002.0 subdirectory contains the default user data table space.

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

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

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

Related concepts:

- “Comparison of SMS and DMS table spaces” in the *Administration Guide: Planning*
- “DMS device considerations” on page 307
- “SMS table spaces” on page 19
- “DMS table spaces” on page 20
- “Illustration of the DMS table-space address map” on page 22

- "Understanding the Recovery History File" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

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

Table space overview

The following sections describe table spaces and discuss the two types of table spaces available in DB2, comparing their advantages and disadvantages.

Table spaces

Two types of table spaces are supported: System Managed Space (SMS) and Database Managed Space (DMS). Each has characteristics that make it appropriate for different environments.

SMS table spaces are an excellent choice for general purposes. They provide good performance with little administration cost. DMS table spaces are the choice for better performance with somewhat more administration cost. DMS table spaces in device containers provide the best performance because double buffering does not occur. Double buffering, which occurs when data is buffered first at the database manager level and then at the file system level, might be an additional cost for file containers or SMS table spaces.

Related concepts:

- "SMS table spaces" on page 19
- "DMS table spaces" on page 20

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 by all containers. The file extension denotes the type of the data stored in the file. To spread the space use evenly across all containers in the table space, the starting extents for tables are placed in round-robin fashion across all containers. Such distribution of extents is particularly important if the database contains many small tables.

Note: SMS table spaces can take advantage of file-system prefetching and caching

In a SMS table space, space for tables is allocated on demand. This expansion is done a single page at a time by default. However, in certain work loads

such as bulk inserts you can improve performance with 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. When you run *db2empfa*, the *multipage_alloc* database configuration parameter is set to Yes. For more information on this tool, refer to the *Command Reference*.

When all space in a single container in an SMS table space is allocated to tables, the table space is considered full, even if space remains in other containers. You can add containers to an SMS table space only on a partition that does not yet have any containers.

Related concepts:

- "Table space design" in the *Administration Guide: Planning*
- "Comparison of SMS and DMS table spaces" in the *Administration Guide: Planning*

Related tasks:

- "Adding a container to an SMS table space on a partition" in the *Administration Guide: Implementation*

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.

Note: Like SMS table spaces, DMS file containers can take advantage of file-system prefetching and caching. However, DMS table spaces cannot.

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 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. When you create a table space, the extent size is defined as a number of contiguous pages. The extent is the unit of space allocation within a table space. Only one table or other object, such as an index, can use the pages in any single extent. All objects created in the table space are allocated extents in a logical table space address map. 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.

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.

Related concepts:

- “Table space design” in the *Administration Guide: Planning*

- "Comparison of SMS and DMS table spaces" in the *Administration Guide: Planning*
- "DMS device considerations" on page 307
- "Database directories and files" on page 16
- "SMS table spaces" on page 19
- "Illustration of the DMS table-space address map" on page 22

Related tasks:

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

Related reference:

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

Illustration of the DMS table-space address map

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

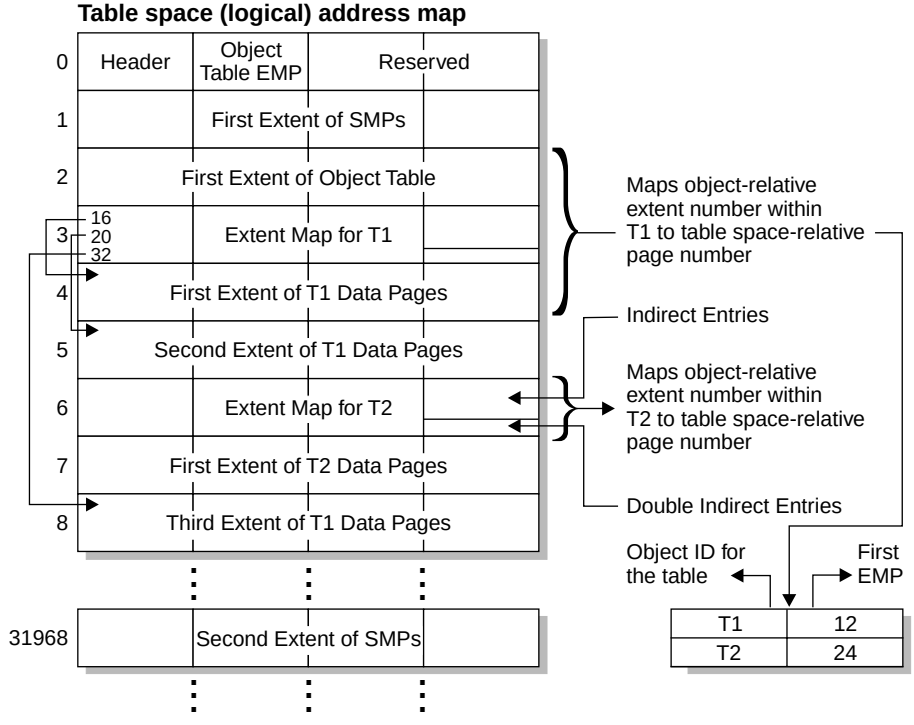


Figure 3. DMS table spaces

The object table is an internal relational table that maps an object identifier to the location of the first EMP extent in the table. 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 round-robin order across the containers associated with the table space.

Related concepts:

- "DMS device considerations" on page 307
- "Disk-storage performance factors" on page 15
- "DMS table spaces" on page 20

Tables and indexes

The following sections discuss management of both standard and MDC tables, and indexes on these tables.

Table and index management for standard tables

DB2® provides two kinds of tables:

- Standard tables, in which the highest logical structure under the table is the row
- Multi-dimensional clustering (MDC) tables, in which the highest logical structure under the table is the block, which is an extent-sized set of consecutive pages

Understanding how tables and indexes are organized can help you understand how you might tune their use. This section describes the logical organization of standard tables and indexes.

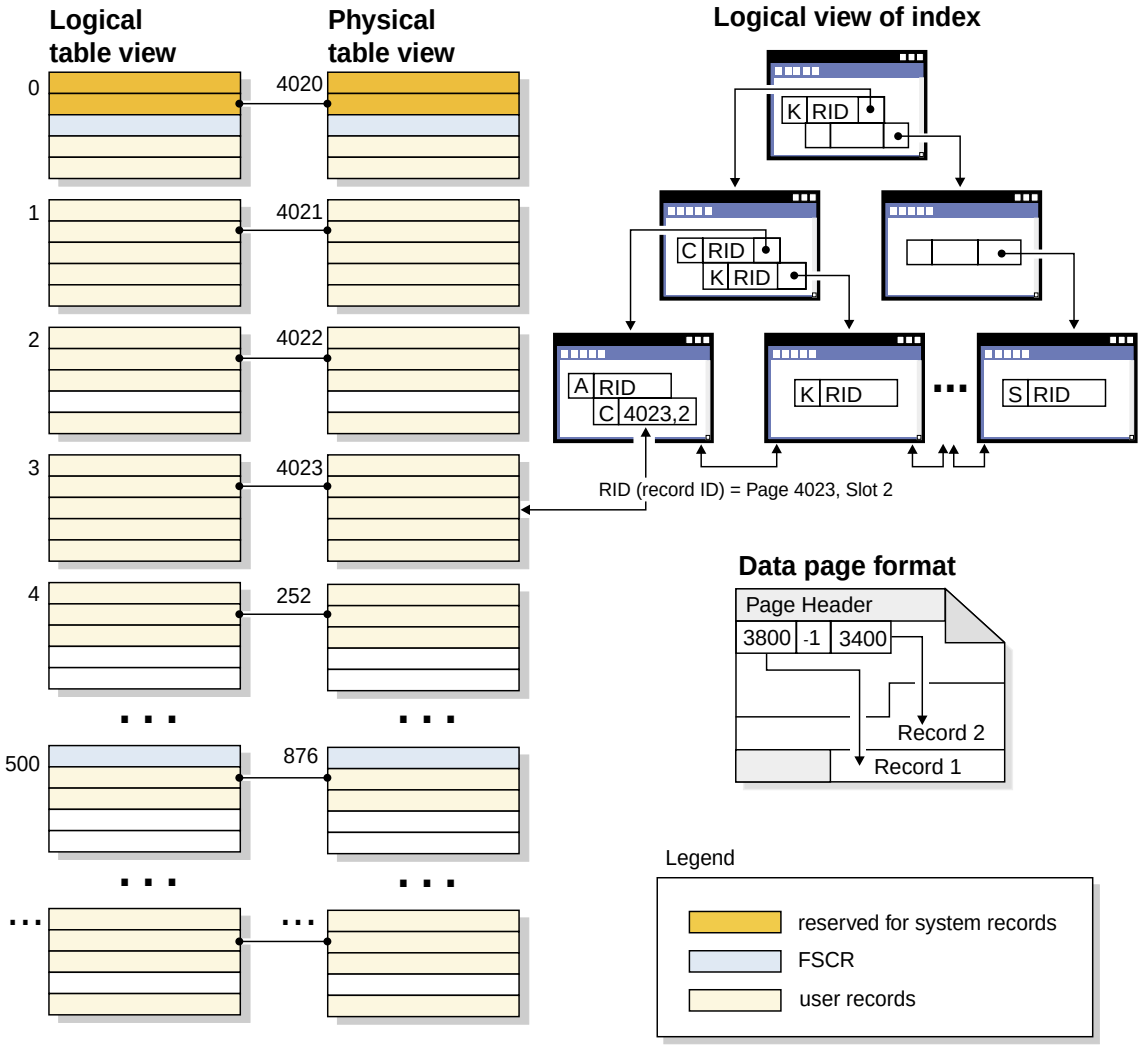


Figure 4. Logical table, record, and index structure for standard tables

Logically, table data is organized as a list of data pages. 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, in a standard table there is a Free Space

Control Record (FSCR) on every 500th data page. These records map the free space for new records 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 that 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. Type-2 index records also contain an additional byte called the ridFlag. The ridFlag stores information about the status of keys in the index, such as whether this key has been marked deleted. 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

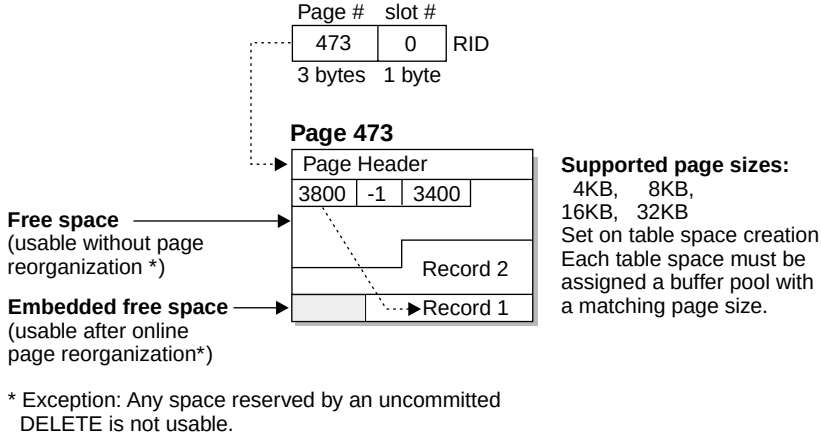


Figure 5. Data page and record-id (RID) format

When a table page is reorganized, embedded free space that is left on the page after a record is physically deleted 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.

Index management in standard tables

DB2 indexes use 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. Index page are usually split 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.

If you migrate from previous versions of DB2, both type-1 and type-2 indexes are in use until you reorganize indexes or perform other actions that convert type-1 indexes to type-2. The index type determines how deleted keys are physically removed from the index pages.

- For type-1 indexes, keys are removed from the index pages during key deletion and index pages are freed when the last index key on the page is removed.
- For type-2 indexes, index keys are removed from the page during key deletion only if there is an X lock on the table. If keys cannot be removed immediately, they are marked deleted and physically removed later. For more information, refer to the section that describes type-2 indexes.

If you have enabled online index defragmentation by setting the MINPCTUSED clause to a value greater than zero when you create the index, index leaf pages can be merged online. The value that you specify is the threshold for the minimum percentage of space used on the index leaf pages. After a key is removed from an index page, if the percentage of space used on the page is at or below the value given, then the database manager attempts 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. Online index defragmentation can improve space reuse, but if the MINPCTUSED value is too high then the time taken to attempt a merge increases but becomes less likely to succeed. The recommended value for this clause is fifty percent or less.

Note: Because online defragmentation occurs only when keys are removed from an index page, in a type-2 index it does not occur if keys are merely marked deleted, but have not been physically removed from the page.

The INCLUDE clause of the CREATE INDEX statement allows the inclusion of a specified column or columns 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. The maintenance cost of updating include columns is less than that of updating key columns, but more than that of updating columns that do not appear in the index. Ordering the index B-tree is only done using the key columns and not the included columns.

Related concepts:

- "Space requirements for database objects" in the *Administration Guide: Planning*
- "Considerations when choosing table spaces for your tables" in the *Administration Guide: Planning*
- "Considerations when choosing MDC table dimensions" in the *Administration Guide: Planning*
- "Table and index management for MDC tables" on page 28

- "Considerations when creating MDC tables" in the *Administration Guide: Planning*
- "Index cleanup and maintenance" on page 302
- "Insert process" on page 34

Table and index management for MDC tables

Table and index organization for multi-dimensional clustering (MDC) tables is based on the same logical structures as standard table organization. Like standard tables, MDC tables are organized into pages that contain rows of data, divided into columns, and the rows on each page are identified by row IDs (RIDs). In addition, however, the pages of MDC tables are grouped into extent-sized blocks. For example, in the illustration below, which shows a table with an extent size of four, the first four pages, numbered 0 through 3, are the first block in the table. The next set of pages, numbered 4 through 7, are the second block in the table.

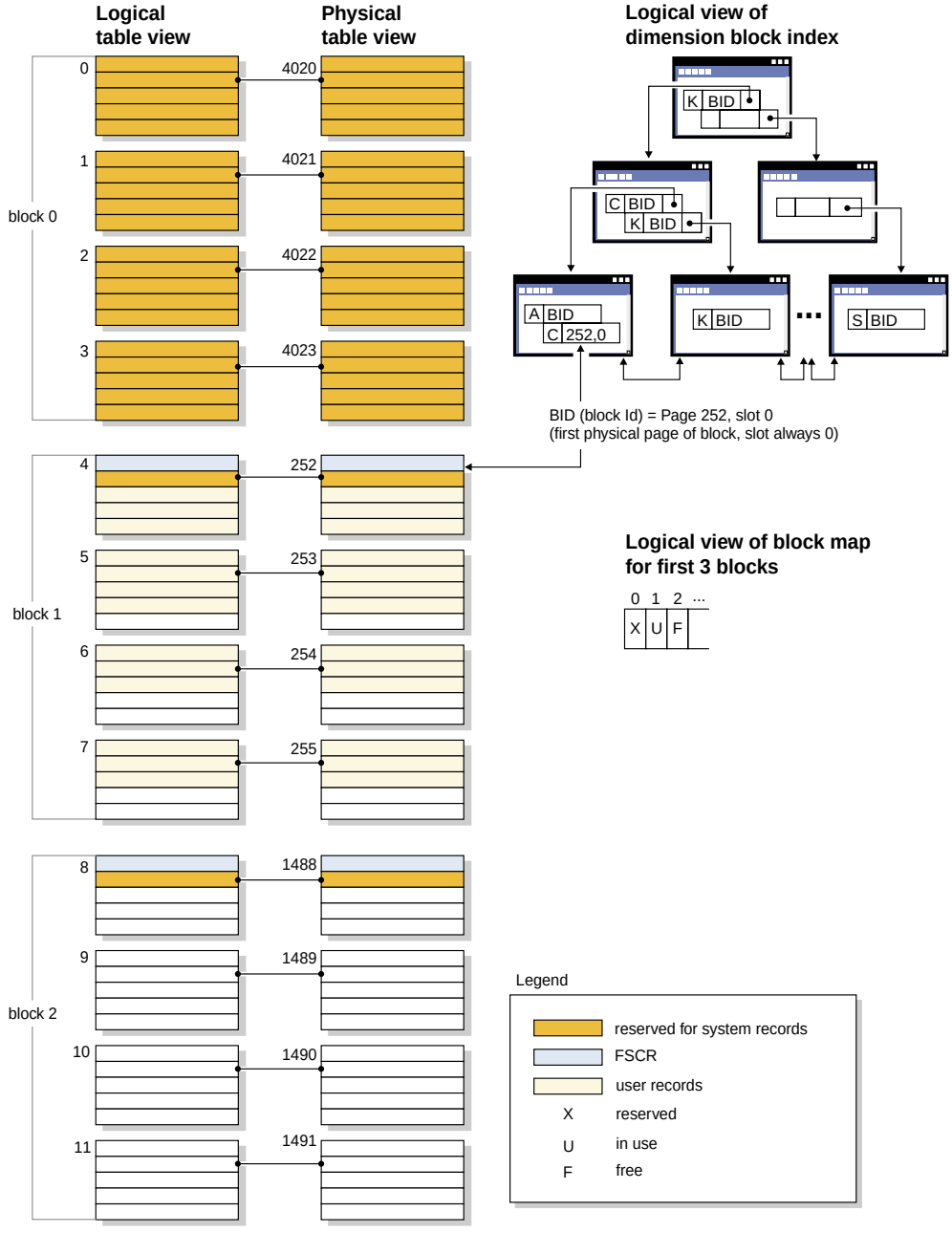


Figure 6. Logical table, record, and index structure for MDC tables

The first block contains special internal records that are used by DB2® to manage the table, including the free-space control record (FSCR). In subsequent blocks, the first page contains the FSCR. An FSCR maps the free

space for new records that exists on each of the pages in the block. This available free space is used when inserting records into the table.

As the name implies, MDC tables cluster data on more than one dimension. Each dimension is determined by a column or set of columns that you specify in the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. When you create an MDC table, the following two kinds of indexes are created automatically:

- A dimension-block index, which contains pointers to each occupied block for a single dimension.
- A composite block index, which contains all dimension key columns. The composite block index is used to maintain clustering during insert and update activity.

The optimizer considers dimension-block index scan plans when it determines the most efficient access plan for a particular query. When queries have predicates on dimension values, the optimizer can use the dimension block index to identify, and fetch from, only extents that contain these values. In addition, because extents are physically contiguous pages on disk, this results in more efficient performance and minimizes I/O.

In addition, you can create specific RID indexes if analysis of data access plans indicates that such indexes would improve query performance.

In addition to the dimension block indexes and the composite block index, MDC tables maintain a block map that contains a bitmap that indicates the availability status of each block. The following attributes are coded in the bitmap list:

- X (reserved): the first block contains only system information for the table.
- U (in use): this block is used and associated with a dimension block index
- L (loaded): this block has been loaded by a current load operation
- C (check constraint): this block is set by the load operation to specify incremental constraint checking during the load.
- T (refresh table): this block is set by the load operation to specify that AST maintenance is required.
- F (free): If no other attribute is set, the block is considered free.

Because each block has an entry in the block map file, the file grows as the table grows. This file is stored as a separate object. In an SMS tablespace it is a new file type. In a DMS table space, it has a new object descriptor in the object table.

Related concepts:

- "Space requirements for database objects" in the *Administration Guide: Planning*

- "Considerations when choosing MDC table dimensions" in the *Administration Guide: Planning*
- "Considerations when creating MDC tables" in the *Administration Guide: Planning*
- "Insert process" on page 34

Index structure

The database manager uses a B+ tree structure for index storage. A B+ tree has one or more levels, as shown in the following diagram, in which RID means row ID:

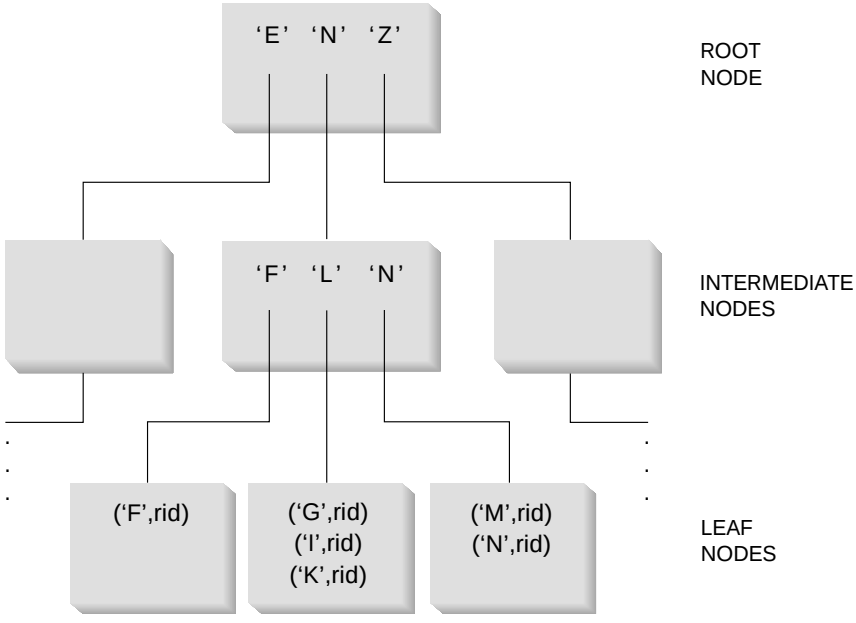


Figure 7. B+ Tree Structure

The top level is called the *root node.* The bottom level consists of *leaf nodes* in which the index key values are stored with pointers to the row in the table that contains the key value. Levels between the root and leaf node levels are called *intermediate nodes.*

When it looks for a particular index key value, the 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 the figure, then to find an index key value, the index manager searches the root node for the first key value greater than or equal to the key being looked for. The root node key points to a specific

intermediate node. The index manager follows this procedure through the intermediate nodes until it finds the leaf node that contains the index key that it needs.

The figure shows the key being looked for as “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” and its corresponding row ID is found. The row ID identifies the corresponding row in the base table. The leaf node level can also contain pointers to previous leaf nodes. These pointers allow the index manager to scan across leaf nodes in either direction to retrieve a range of values after it finds one value in the range. The ability to scan in either direction is only possible if the index was created with the ALLOW REVERSE SCANS clause.

For multi-dimensional clustering (MDC) tables, a block index is created automatically for each clustering dimension that you specify for the table. An additional composite block index is also created, which contains a key part for each column involved in any dimension of the table. These indexes contain pointers to block IDs (BIDs) instead of RIDs, and provide data-access improvements.

In DB2® Version 8.1 and later, indexes can be of either type 1 or type 2. A *type-1 index* is the older index style. Indexes that you created in earlier versions of DB2 are of this kind.

A type-2 index is somewhat larger than a type-1 index and provides features that minimize next-key locking. The one-byte *ridFlag* byte stored for each RID on the leaf page of a type-2 index is used to mark the RID as logically deleted so that it can be physically removed later. For each variable length column included in the index, one additional byte stores the actual length of the column value. Type-2 indexes might also be larger than type-1 indexes because some keys might be marked deleted but not yet physically removed from the index page. After the DELETE or UPDATE transaction is committed, the keys marked deleted can be cleaned up.

Related concepts:

- “Advantages and disadvantages of indexes” on page 294
- “Index reorganization” on page 303
- “Online index defragmentation” on page 305

Processes

The following sections provide general descriptions of the DB2 processes.

Logging process

All databases maintain log files that keep records of database changes. There are two logging strategy choices:

- Circular logging, in which 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, in which a log file is archived when it fills with log records. 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.

Regardless of the logging strategy, all changes to regular data and index pages are written to the log buffer. The data in the log buffer is written to disk by the logger process. In the following circumstances, query processing must wait for log data to be written to disk:

- On COMMIT
- Before the corresponding data pages are written to disk, because DB2® uses write-ahead logging. The benefit of write-ahead logging is that when a transaction completes by executing the COMMIT statement, not all of the changed data and index pages need to be written to disk.
- Before some changes are made to metadata, most of which result from executing DDL statements
- On writing log records into the log buffer, if the log buffer is full

DB2 manages writing log data to disk in this way in order to minimize processing delay. In an environment in which many short concurrent transactions occur, most of the processing delay is caused by COMMIT statements that must wait for log data to be written to disk. As a result, the logger process frequently writes small amounts of log data to disk, with additional delay caused by log I/O overhead. To balance application response time against such logging delay, set the *mincommit* database configuration parameter to a value greater than 1. This setting might cause longer delay for COMMIT from some applications, but more log data might be written in one operation.

Changes to large objects (LOBs) and LONG VARCHARs are tracked through shadow paging. LOB column changes are not logged unless you specify log retain and the LOB column is defined on the CREATE TABLE statement without the NOT LOGGED clause. Changes to allocation pages for LONG or LOB data types are logged like regular data pages.

Related concepts:

- "Update process" on page 35
- "Client-server processing model" on page 36

Related reference:

- "Number of Commits to Group configuration parameter - mincommit" on page 464

Insert process

When SQL statements use INSERT to place new information in a table, an INSERT search algorithm first searches the Free Space Control Records (FSCRs) to find a page with enough space. However, even when the FSCR indicates a page has enough free space, the space may not be usable because it is reserved by an uncommitted DELETE from another transaction. To ensure that uncommitted free space is usable, you should COMMIT transactions frequently.

The setting of the DB2MAXFSCRSEARCH registry variable determines the number of FSCRs in a table that are searched for an INSERT. The default value for this registry variable is five. If no space is found within the specified number of FSCRs, the inserted record is appended at the end of the table. To optimize INSERT speed, subsequent records are also appended to the end of the table until two extents are filled. After the two extents are filled, the next INSERT resumes searching at the FSCR where the last search ended.

Note: To optimize for INSERT speed at the possible expense of faster table growth, set the DB2MAXFSCRSEARCH registry variable to a small number. To optimize for space reuse at the possible expense of INSERT speed, set DB2MAXFSCRSEARCH to a larger number.

After all FSCRs in the entire table have been searched in this way, the records to be inserted are appended without additional searching. Searching using the FSCRs is not done again until space is created somewhere in the table, such as following a DELETE.

There are two other INSERT algorithm options, as follows:

- APPEND MODE

  In this mode, new rows are always appended to the end of the table. No searching or maintenance of FSCRs takes place. This option is enabled using the ALTER TABLE APPEND ON statement, and can improve performance for tables that only grow, like journals.
- A clustering index is defined 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, except that a worst-fit approach is used instead of a first-fit approach. This worst-fit approach tends to choose pages with more free space. This method establishes a new clustering area for rows with this key value.

When you define a clustering index on a table, use ALTER TABLE... PCTFREE before you either load or reorganize the table. The PCTFREE clause specifies the percentage of free space that should remain on the data page of the table after loading and reorganizing. This increases the probability that the cluster index operation will find free space on the appropriate page.

Related concepts:

- "Table and index management for standard tables" on page 23
- "Update process" on page 35
- "Table and index management for MDC tables" on page 28

Update process

When an agent updates a page, the database manager uses the following protocol to minimize the I/O required by the transaction and ensure recoverability.

1. 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.
2. The change is made to the page.
3. The page is unlatched and unfixed.

   The page is considered to be "dirty" because changes to the page have not been written out to disk.
4. The log buffer is updated.

   Both the data in the log buffer and the "dirty" data page are forced to disk.

For better performance, these I/Os are delayed until a convenient point, such as during a lull in the system load, or until necessary to ensure recoverability, or to limit recovery time. Specifically, a "dirty" page is forced to disk at the following times:

- 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. When this value is 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 statement is issued. When the COMMIT statement is executed, all changed pages are flushed to disk to ensure recoverability.

Related concepts:

- "Logging process" on page 33
- "Client-server processing model" on page 36

Related reference:

- "Recovery Range and Soft Checkpoint Interval configuration parameter - softmax" on page 465
- "Changed Pages Threshold configuration parameter - chngpgs_thresh" on page 431

Client-server processing 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.

Note: How DB2® manages client connections depends on whether the connection concentrator is on or off. The connection concentrator is ON when the *max_connections* database manager configuration parameter is set larger than the *max_coordagents* configuration parameter.

- If the connection concentrator is OFF, each client application is assigned a unique EDU called a *coordinator agent* that coordinates the processing for that application and communicates with it.
- If the connection concentrator is ON, each coordinator agent can manage many client connections, one at a time, and might coordinate the other worker agents to do this work. For Internet applications with many relatively transient connections, or similar applications with many relatively small transactions, the connection concentrator improves performance by allowing many more client applications to be connected. It also reduces system resource use for each connection.

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.

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.

At A1 in the figure below, a local client establishes communications first through the db2ipccm. At A2, the db2ipccm works with a db2agent EDU, which becomes the coordinator agent for the application requests from the local client. The coordinator agent then contacts the client application at A3 to establish shared memory communications between the client application and the coordinator. The application at the local client is connected to the database at A4.

At B1 in the figure below, a remote client establishes communications through the db2tcpcm EDU. If any other communications protocol is chosen, the appropriate communication manager is used. The db2tcpcm EDU establishes TCP/IP communication between the client application and the db2tcpcm. It then works with a db2agent at B2, which becomes the coordinator agent for the application and passes the connection to this agent. At B3 the coordinator agent contacts the remote client application and is connected to the database.

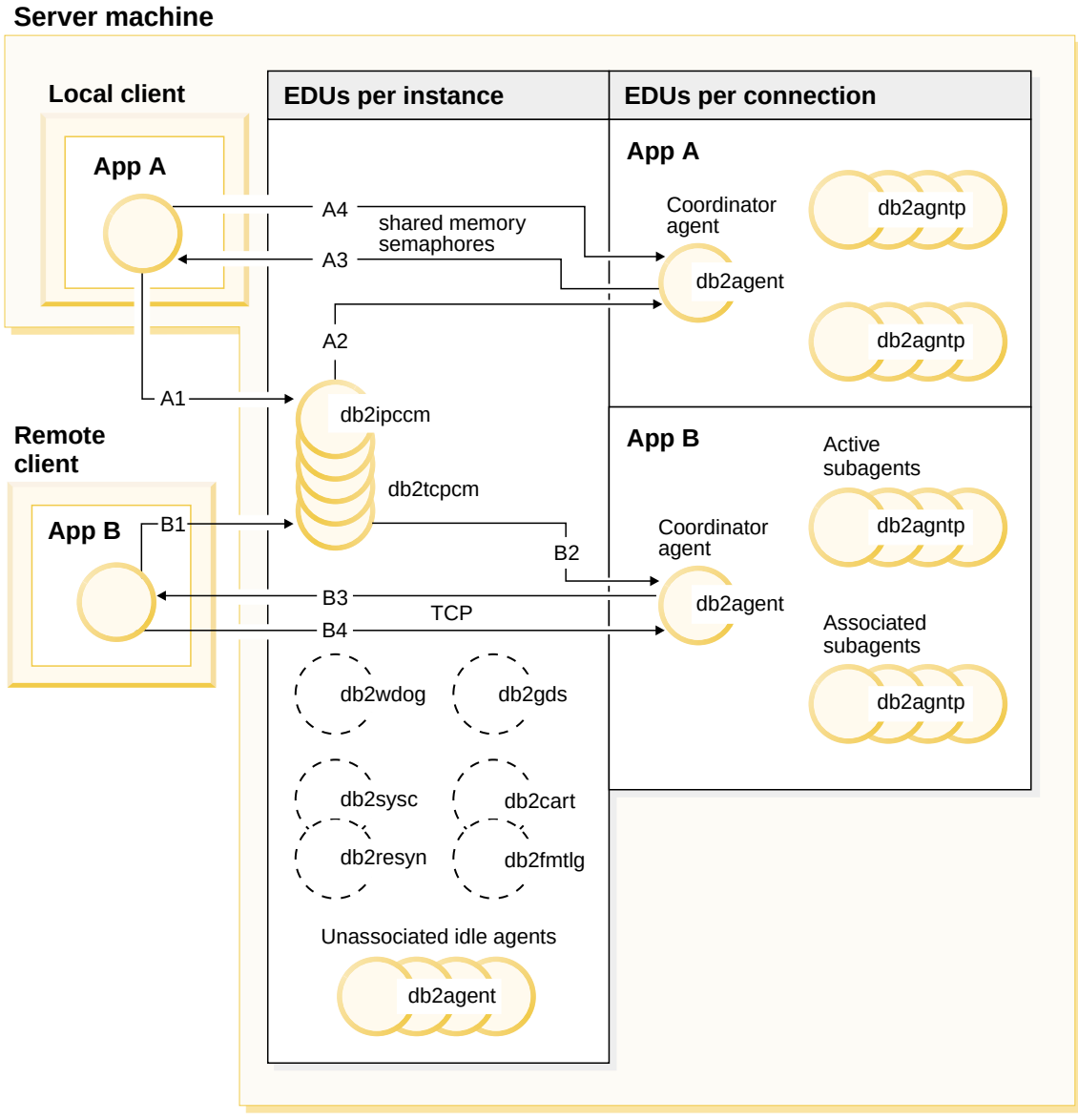


Figure 8. Process model overview

Other things to notice in this figure:

- Worker agents carry out application requests.
- There are four types of worker agents: active coordinator agents, active subagents, associated subagents, and idle agents.
- Each client connection is linked to an active coordinator agent.

- In a partitioned database environment, and enabled intra-partition 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 and pooled agents wait for new work.
- Other EDUs manage client connections, logs, two-phase COMMITs, backup and restore tasks, and other tasks.

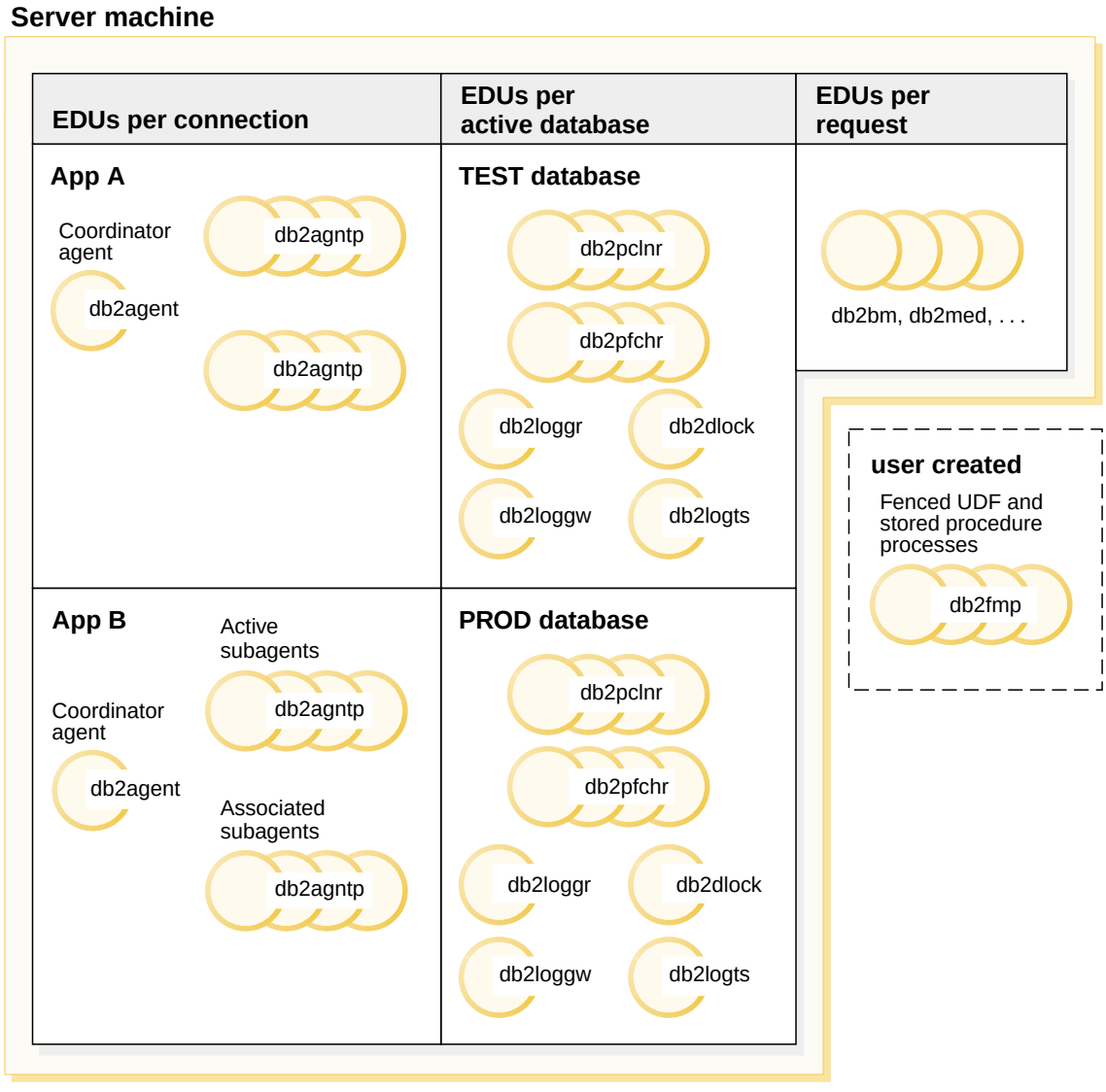


Figure 9. Process model, part 2

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).

Fenced user-defined functions (UDFs) and stored procedures, which are not shown in the figure, are managed to minimize costs associated with their creation and destruction. The default for the *keepfenced* database manager

configuration parameter is “YES”, which keeps the stored procedure process available for re-use at the next stored procedure call.

Note: Unfenced UDFs and stored procedures run directly in an agent’s address space for better performance. However, because they have unrestricted access to the agent’s address space, they need to be rigorously tested before being used.

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 shows the similarities and differences between the single partition processing model as seen in the previous two figures, and the multiple partition processing model.

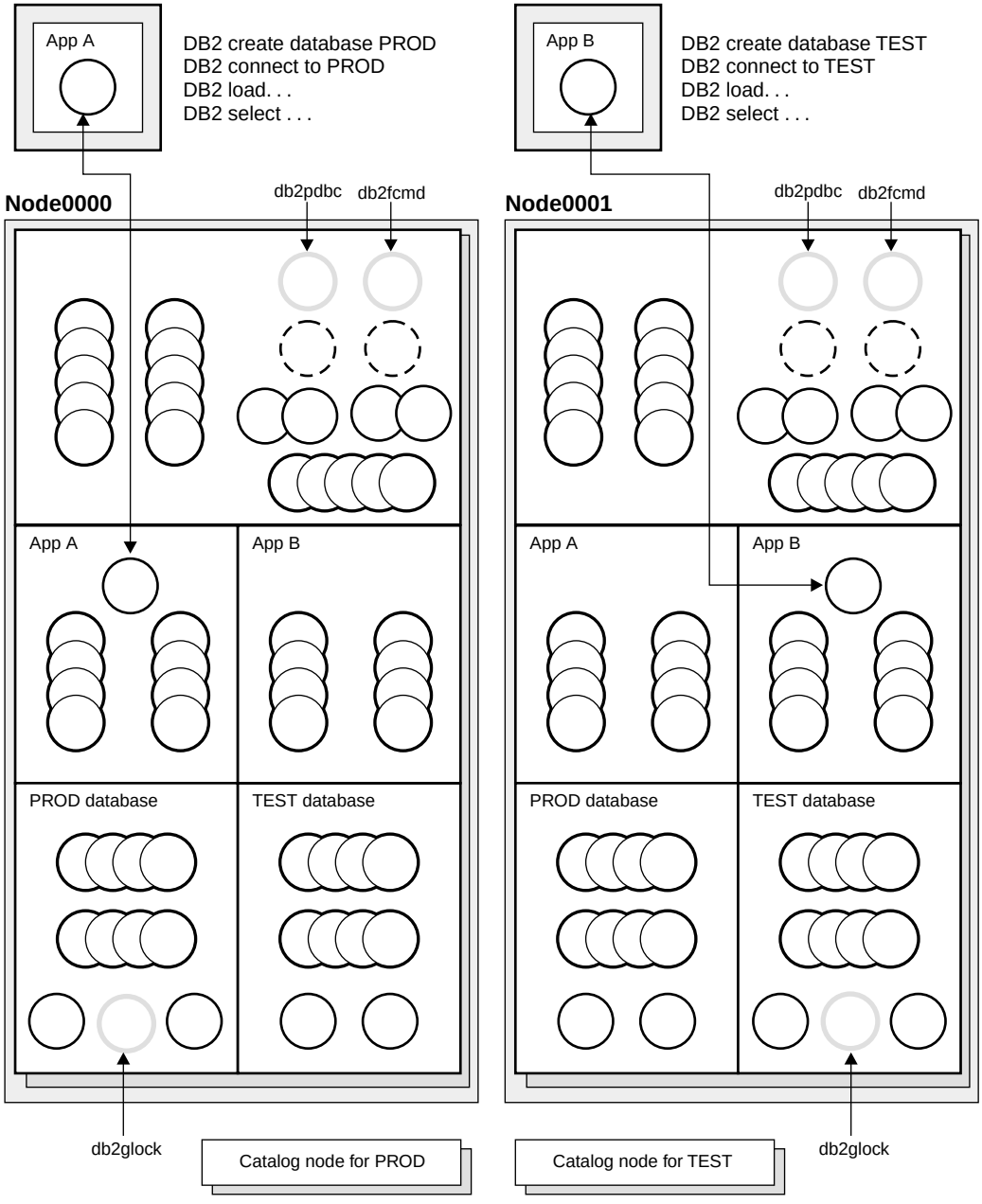


Figure 10. Process model and multiple partitions

Most 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 is 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 might want to create your databases on different nodes 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 deadlocks across the nodes where the active database is located.

Each CONNECT from an application is represented by a connection that is associated with a coordinator agent to handle the connection. The *coordinator agent* is the agent that communicates with the application, receiving requests and sending replies. It can either satisfy the request itself or coordinate multiple subagents to work on the request. The partition where the coordinator agent exists is called the *coordinator node* of that application. The coordinator node can also be set with the SET CLIENT CONNECT_NODE command.

Parts of the database requests from the application are sent 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.

Related concepts:

- "DB2 architecture and process overview" on page 11
- "Logging process" on page 33
- "Update process" on page 35
- "Memory management" on page 44
- "Connection-concentrator improvements for client connections" on page 312

Memory management

A primary performance tuning task is deciding how to divide the available memory among the areas within the database. You tune this division of memory by setting the key configuration parameters described in this section.

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 Database Shared Memory of that database. 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. All other memory is attached or allocated from the instance shared memory. If the 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. Instance shared memory can be controlled by the *instance_memory* configuration parameter. By default, this parameter is set to `automatic` so that DB2® calculates the amount of memory allocated for the instance.

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. Database Shared Memory can be controlled by the *database_memory* configuration parameter. By default, this parameter is set to *automatic* so that DB2 calculates the amount of memory allocated for 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 .
- Utility heap
- Package cache
- Catalog cache

Note: Memory can be allocated, freed, and exchanged between different areas while the database is running. For example, you can decrease the catalog cache and then increase any given bufferpool by the same amount. However, before changing the configuration parameter dynamically, you must be connected to that database. All the memory areas listed above can be changed dynamically.

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 only in a partitioned database environment, or in a non-partitioned database with intra-parallelism enabled, or if the connection concentrator is enabled. This memory is used by agents that do the work requested by clients connected to the database.

The database manager configuration parameter *max_connections* sets an upper limit to the number of applications that connect to a database. Since each application that attaches to a database causes some 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 partitioned 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 created. The agent private memory 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.

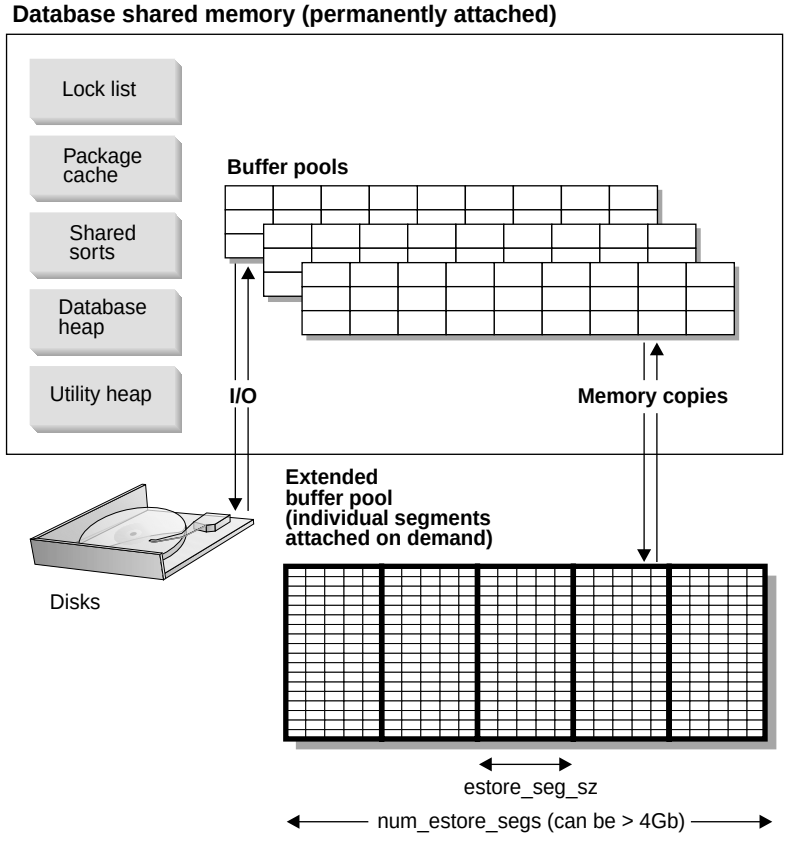


Figure 11. How extended storage is used by buffer pools

Extended storage acts as an extended look-aside buffer for the main buffer pools. It can be much larger than 4 GB. For 32-bit computers with large amounts of main memory, look-aside buffers can exploit such memory performance improvements. The extended storage cache is defined in terms of memory segments. For 64-bit computers, such methods are not needed to access all available memory.

Note, however, that if you 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 journaled file-system cache or as a 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.

Related concepts:

- "Organization of memory use" on page 251
- "Database manager shared memory" on page 254
- "Global memory and parameters that control it" on page 258
- "Buffer-pool management" on page 264
- "Secondary buffer pools in extended memory on 32-bit platforms" on page 266
- "Guidelines for tuning parameters that affect memory usage" on page 261
- "Connection-concentrator improvements for client connections" on page 312

Related reference:

- "Extended Storage Memory Segment Size configuration parameter - estore_seg_sz" on page 437
- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Number of Extended Storage Memory Segments configuration parameter - num_estore_segs" on page 437
- "Maximum Number of Agents configuration parameter - maxagents" on page 444
- "Maximum Number of Concurrently Active Databases configuration parameter - numdb" on page 514
- "Maximum Number of Client Connections configuration parameter - max_connections" on page 448
- "Instance Memory configuration parameter - instance_memory" on page 421
- "Database Shared Memory Size configuration parameter - database_memory" on page 391

Part 2. Tuning application performance

Chapter 3. Application considerations

A number of factors can impact the run-time performance of your application. This chapter describes some of the factors to consider when you design and code your application, and later when you tune its performance.

Concurrency control and isolation levels

The following sections describe the effect of different isolation levels on concurrency.

Concurrency issues

Because many users access and change data in a relational database, the database manager must be able both to allow users to make these changes and to ensure that data integrity is preserved. *Concurrency* refers to 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).

Some additional ("phantom") rows are returned as part of the result set that were not returned when the query was initially executed (step 1).

Note: Declared temporary tables have no concurrency issues because they are available only 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.

Concurrency control in federated database systems

A *federated database system* supports applications and users submitting SQL statements that reference two or more database management systems (DBMSs) or databases in a single statement. To reference the data sources, which consist of a DBMS and data, DB2® uses *nicknames*. Nicknames are aliases for objects in other database managers. In a federated system, DB2 relies on the concurrency control protocols of the database manager that hosts the requested data.

A DB2 federated system provides *location transparency* for database objects. For example, with location transparency if information about tables and views is moved, references to that information through nicknames can be updated without changing applications that request the information. When an application accesses data through nicknames, DB2 relies on the concurrency control protocols of data-source database managers to ensure isolation levels. Although DB2 tries to match the requested level of isolation at the data source with a logical equivalent, results may vary depending on data source capabilities.

Related concepts:

- "Performance impact of isolation levels" on page 52

Related tasks:

- "Specifying the isolation level" on page 57

Related reference:

- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428

Performance impact of isolation levels

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

with a DECLARE CURSOR statement 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. DB2® supports the following isolation levels:

- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read.

Note: Some host database servers support the *no commit* isolation level. On other databases, this isolation level behaves like the uncommitted read isolation level.

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. For example, 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. In order to avoid lock escalation, the

optimizer may elect to acquire a single table-level lock immediately for an index scan, if it believes that lock escalation is very likely to occur. 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.

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.

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.

When it runs a program using isolation level UR, an application can use isolation level CS. This happens because the cursors used in the application program are ambiguous. The ambiguous cursors can be escalated to isolation level CS because of a BLOCKING option. The default for the BLOCKING option is UNAMBIG. This means that ambiguous cursors are treated as updatable and the escalation of the isolation level to CS occurs. To prevent this escalation, you have the following two choices:

- Modify the cursors in the application program so that they are unambiguous. Change the SELECT statements to include the FOR READ ONLY clause.
- Leave cursors ambiguous in the application program, but precompile the program or bind it with the BLOCKING ALL option to allow any ambiguous cursors to be treated as read-only when the program is run.

As in the example given for Repeatable Read, 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 executing only select statements and you do not care whether you see uncommitted data from other applications.

Summary of isolation levels

The following table summarizes the different isolation levels in terms of their undesirable effects.

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

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

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 that are required to obtain and free locks vary with the isolation level. The potential for deadlock situations also varies with the isolation level.

Related concepts:

- "Concurrency issues" on page 51

Related tasks:

- "Specifying the isolation level" on page 57

Specifying the isolation level

Because the isolation level determines how data is locked and isolated from other processes while the data is being accessed, you should select an isolation level that balances the requirements of concurrency and data integrity. The isolation level that you specify is in effect for the duration of the unit of work.

Note: Many commercially written applications provide a method for choosing the isolation level. Refer to the application documentation for information.

Procedure:

To specify the isolation level:

1. **At precompile or bind time:**

   For an application written in a supported compiled language, use the ISOLATION option of the command line processor PREP or BIND commands. You can also use the PREP or BIND APIs to specify the isolation level.

   - If you create a bind file at precompile time, the isolation level is stored in the bind file. If you do not specify an isolation level at bind time, the default is the isolation level used during precompilation.
   - If you do not specify an isolation level, the default of cursor stability is used.

   Note: To determine the isolation level of a package, execute 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.

2. **On database servers that support REXX:**

   When a database is created, multiple bind files that support the different isolation levels for SQL in REXX are bound to the database. Other command-line processor packages are also bound to the database when a database is created.

   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.

   To verify the isolation level in use by a REXX application, check the value of the SQLISL REXX variable. The value is updated every time the CHANGE SQLISL command is executed.

3. **At the statement level:**

   Use the WITH clause. The statement-level isolation level overrides the isolation level specified for the package in which the statement appears.

   You can specify an isolation level for the following SQL statements:

   - SELECT
   - SELECT INTO
   - Searched DELETE
   - INSERT
   - Searched UPDATE
   - DECLARE CURSOR

   The following conditions apply to isolation levels specified for statements:

   - The WITH clause cannot be used on subqueries
   - The WITH UR option applies only to read-only operations. In other cases, the statement is automatically changed from UR to CS.

4. **From CLI or ODBC at runtime:**

   Use the CHANGE ISOLATION LEVEL command. For DB2 Call Level Interface (DB2 CLI), you can change the isolation level as part of the DB2 CLI configuration. At runtime, use the *SQLSetConnectAttr* function with the SQL_ATTR_TXN_ISOLATION attribute to set the transaction isolation level for the current connection referenced by the *ConnectionHandle*. You can also use the TXNISOLATION keyword in the db2cli.ini file .

5. **When working with JDBC or SQLJ at run time:**

Note: JDBC and SQLJ are implemented with CLI on DB2, which means the db2cli.ini settings might affect what is written and run using JDBC and SQLJ.

Use the setTransactionIsolation method in the java.sql interface connection.

In SQLJ, you run the db2profc SQLJ optimizer to create a package. The options that you can specify for this package include its isolation level.

Related concepts:

- "Concurrency issues" on page 51

Related reference:

- "SQLSetConnectAttr Function (CLI) - Set Connection Attributes" in the *CLI Guide and Reference, Volume 2*
- "CONNECT (Type 1) statement" in the *SQL Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Concurrency control and locking

The following sections describe the different kinds and levels of locking and how these locks are determined by and affect data access performance.

Locks and concurrency control

To provide concurrency control and prevent uncontrolled data access, the database manager places locks on tables, table blocks, or table rows. A *lock* associates a database manager resource with an application, called the *lock owner*, to control how other applications can access the same resource.

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 the following:
  - Uncommitted Read (UR)
  - Cursor Stability (CS)
  - Read Stability (RS)
  - Repeatable Read (RR)

  The different isolation levels are used to control access to uncommitted data, prevent lost updates, allow non-repeatable reads of data, and prevent 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 LOCKSIZE attribute for the table. 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. If lock escalations occur frequently, increase the value of either `locklist` or `maxlocks`, or both.

Ensure that 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 LOCKSIZE attribute is “TABLE”.
- The lock list fills, causing escalation.
- 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* occurs when the number of locks held on rows and tables in the database equals the percentage of the locklist specified by the `maxlocks` database configuration parameter. Lock escalation might not affect the table that acquires the lock that triggers escalation. To reduce the number of locks to about half the number held when lock escalation begins, the database manager begins converting many small row locks to table locks for all active tables, beginning with any locks on large object (LOB) or long VARCHAR elements. An *exclusive lock escalation* is a lock escalation in which the table lock acquired is an exclusive lock. Lock escalations reduce concurrency. Conditions that might cause lock escalations should be avoided.

The duration of row locking varies with the isolation level being used:

- UR scans: No row locks are held unless row data is changing.
- CS scans: Row locks are only held while the cursor is positioned on the row.

- RS scans: Only qualifying row locks are held for the duration of the transaction.
- RR scans: All row locks are held for the duration of the transaction.

Related concepts:

- "Lock attributes" on page 61
- "Locks and performance" on page 63
- "Guidelines for locking" on page 68

Related reference:

- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Time Interval for Checking Deadlock configuration parameter - dlchktime" on page 427
- "Diagnostic Error Capture Level configuration parameter - diaglevel" on page 507
- "Lock Timeout configuration parameter - locktimeout" on page 430
- "Lock type compatibility" on page 72
- "Lock modes and access paths for standard tables" on page 74
- "Lock modes for table and RID index scans of MDC tables" on page 77
- "Locking for block index scans for MDC tables" on page 80

Lock attributes

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 type of object that you can lock explicitly is a table. The database manager also imposes locks on other types of resources, such as rows, tables, and table spaces. For multidimensional clustering (MDC) tables, block locks can also be imposed. The object being locked determines the *granularity* of the lock.

Duration
The length of time a lock is held. The isolation level in which the query runs affects the lock duration.

The following table shows the modes and their effects in order of increasing control over resources. For detailed information about locks at various levels, refer to the lock-mode reference tables.

Table 3. Lock Mode Summary

Lock Mode	Applicable Object Type	Description
IN (Intent None)	Table spaces, blocks, tables	The lock owner can read any data in the object, including uncommitted data, but cannot update any of it. Other concurrent applications can read or update the table.
IS (Intent Share)	Table spaces, blocks, tables	The lock owner can read data in the locked table, but cannot update this data. 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 of the application is either RS or CS. NS lock mode is not used for next-key locking. It is used instead of S mode during CS and RS scans to minimize the impact of next-key locking on these scans.
S (Share)	Rows, blocks, tables	The lock owner and all concurrent applications can read, but not update, the locked data.
IX (Intent Exclusive)	Table spaces, blocks, tables	The lock owner and concurrent applications can read and update data. Other concurrent applications can both read and update the table.
SIX (Share with Intent Exclusive)	Tables, blocks	The lock owner can read and update data. Other concurrent applications can read the table.
U (Update)	Rows, blocks, tables	The lock owner can update data. Other units of work can read the data in the locked object, but cannot attempt to update it.
NW (Next Key Weak Exclusive)	Rows	When a row is inserted into an index, an NW lock is acquired on the next row. For type 2 indexes, this occurs only if the next row is currently locked by an RR scan. The lock owner can read but not update the locked row. This lock mode is similar to an X lock, except that it is also compatible with W and NS locks.
X (Exclusive)	Rows, blocks, tables	The lock owner can both read and update data in the locked object. Only uncommitted read applications can access the locked object.
W (Weak Exclusive)	Rows	This lock is acquired on the row when a row is inserted into a table that does not have type-2 indexes defined. The lock owner can change the locked row. To determine if a duplicate value has been committed when a duplicate value is found, this lock is also used during insertion into a unique index. 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.

Table 3. Lock Mode Summary (continued)

Lock Mode	Applicable Object Type	Description
Z (Super Exclusive)	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 for some types of table reorganization. No other concurrent application can read or update the table.

Related concepts:

- "Locks and concurrency control" on page 59
- "Locks and performance" on page 63

Related reference:

- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
- "Lock type compatibility" on page 72
- "Lock modes and access paths for standard tables" on page 74

Locks and performance

Several related factors affect the uses of locks and their effect on application performance. The following factors are discussed here:

- Concurrency and granularity
- Lock compatibility
- Lock conversion
- Lock escalation
- Lock waits and timeouts
- Deadlocks

Concurrency and granularity

If one application holds a lock on a database object, another application might not be able to access that object. For this reason, row-level locks are better for maximum concurrency than table-level locks. However, locks require storage and processing time, so a single table lock minimizes lock overhead.

The LOCKSIZE clause of the ALTER TABLE statement specifies the scope (granularity) of locks at either row or table level. By default, row locks are used. Only S (Shared) and X (Exclusive) locks are requested by these defined table locks. The ALTER TABLE statement LOCKSIZE ROW clause does not prevent normal lock escalation from occurring.

A permanent table lock defined by the ALTER TABLE statement might be preferable to a single-transaction table lock using LOCK TABLE statement in the following cases:

- The table is read-only, and will always need only S locks. Other users can also obtain S locks on the table.
- The table is usually accessed by read-only applications, but is sometimes accessed by a single user for brief maintenance, and that user requires an X lock. While the maintenance program runs, the read-only applications are locked out, but in other circumstances, read-only applications can access the table concurrently with a minimum of locking overhead.

The ALTER TABLE statement specifies locks globally, affecting all applications and users that access that table. Individual applications might use the LOCK TABLE statement to specify table locks at an application level instead.

Lock compatibility

Lock compatibility is another important factor in concurrent access of tables. Lock compatibility refers to the current lock on the object and the type of lock being requested on that object and determines whether the request can be granted.

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

Changing the mode of a lock already held is called a *conversion*. Lock conversion occurs when a process accesses a data object on which it already holds a lock, and the access mode 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 request a lock many times on the same data object indirectly through a query.

Some lock modes apply only to tables, others only to rows or blocks. For rows or blocks, conversion usually occurs if an X is needed and an S or U (Update) lock is held.

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 is required, the resulting conversion is to a SIX (Share with Intent Exclusive) lock. All other conversions result in the requested lock mode becoming the mode of the lock held if the requested mode is more restrictive.

A dual conversion might also occur when a query updates a row. If the row is read through an index access and locked as S, the table that contains the row has a covering intention lock. But if the lock type is IS instead of IX, if the row is subsequently changed the table lock is converted to an IX and the row to an X.

Lock conversion usually takes place implicitly as a query is executed. Understanding the kinds of locks obtained for different queries and table and index combinations can assist you in designing and tuning your application.

Lock Escalation

Lock escalation is an internal mechanism that reduces the number of locks held. In a single table, locks are escalated to a table lock from many row locks, or for multi-dimensional clustering (MDC) tables, from many row or block locks. Lock escalation occurs when applications hold too many locks of any type. Lock escalation can occur for a specific database agent if the agent exceeds its allocation of the lock list. Such escalation is handled internally; the only externally detectable result might be a reduction in concurrent access on one or more tables.

In an appropriately configured database, lock escalation occurs infrequently. For example, lock escalation might occur when an application designer creates an index on a large table to increase performance and concurrency but a transaction accesses most of the records in the table. In this case, because the database manager cannot to predict how much of the table will be locked, it locks each record individually instead of locking only the table either S or X. In this case, the database designer might consult with the application designer, and recommend a LOCK TABLE statement for this transaction.

Occasionally, the process receiving the internal escalation request holds few or no row locks on any table, but locks are escalated because one or more processes hold many locks. The process might not request another lock or access the database again except to end the transaction. Then another process might request the lock or locks that trigger the escalation request.

Note: Lock escalation might also cause deadlocks. For example, suppose a read-only application and an update application are both accessing the same table. If the update application has exclusive locks on many rows

on the table, the database manager might try to escalate the locks on this table to an exclusive table lock. However, the table lock held by the read-only application will cause the exclusive lock escalation request to wait. If the read-only application requires a row lock on a row already locked by the update application, this creates a deadlock. To avoid this kind of problem, either code the update application to lock the table exclusively when it starts or increase the size of the lock list.

Lock waits and timeouts

Lock timeout detection is a database manager feature that prevents applications from waiting indefinitely for a lock to be released in an abnormal situation. For example, a transaction might be waiting for a lock held by another user's application, but the other user has left the workstation without allowing the application to commit the transaction that would release the lock. To avoid stalling an application in such a case, set the *locktimeout* configuration parameter to the maximum time that any application should wait to obtain a lock.

Setting this parameter helps avoid global deadlocks, especially in distributed unit of work (DUOW) applications. If the time that the lock request is pending is greater than the *locktimeout* value, the requesting application receives an error and its 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 period expires. The default value for *locktimeout* is -1, which turns off lock timeout detection.

To log more information about lock-request timeouts in the administration notification log, set the database manager configuration parameter *notifylevel* to four. The logged information includes the locked object, the lock mode, and the application holding the lock. The current dynamic SQL statement or static package name might also be logged. A dynamic SQL statement is logged only at *notifylevel* four.

Deadlocks

Contention for locks can result in deadlocks. For example, suppose that 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 are in a deadlock. In a deadlock, both processes are suspended until their second lock request is granted, but neither request is granted until one of the processes performs a commit or rollback. This state continues indefinitely until an external agent activates one of the processes and forces it to perform a rollback.

To handle deadlocks, the database manager uses an asynchronous system background process called the deadlock detector. The deadlock detector becomes active at intervals specified by the *dlchktime* configuration parameter. When activated, the deadlock detector examines the lock system for deadlocks. In a partitioned database, each partition sends *lock graphs* to the database partition that contains the system catalog views. Global deadlock detection takes place on this partition.

If it finds a deadlock, the deadlock detector selects one deadlocked process as the *victim process* to roll back. The victim process is awakened, and returns SQLCODE -911 (SQLSTATE 40001), with reason code 2, to the calling application. The database manager rolls back the selected process automatically. When the rollback is complete, the locks that belonged to the victim process are released, and the other processes involved in the deadlock can continue.

To ensure good performance, select the proper interval for the deadlock detector. An interval that is too short causes unnecessary overhead, and an interval that is too long allows a deadlock to delay a process for an unacceptable amount of time. For example, a wake-up interval of 5 minutes allows a deadlock to exist for almost 5 minutes, which can seem like a long time for short transaction processing. Balance the possible delays in resolving deadlocks with the overhead of detecting them.

In a partitioned database, the *dlchktime* configuration parameter interval is applied only at the catalog node. If a large number of deadlocks are detected in a partitioned database, increase the value of the *dlchktime* parameter to account for lock waits and communication waits.

A different problem occurs when an application with more than one independent process that accesses the database is structured 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 chance of deadlocks increases because of potential contention between the processes for the same data. For instance, if two processes read the table, and then update the table, process A might try to get an X lock on a row on which process B has an S lock, and vice versa. To avoid such 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 might consider defining a monitor that records when deadlocks occur. Use the SQL statement CREATE EVENT to create a monitor.

In a federated system environment in which an application accesses nicknames, the data requested by the application might not be available because of a deadlock at a data source. When this happens, DB2® relies on the deadlock handling facilities at the data source. If deadlocks occur across more than one data source, DB2 relies on data source timeout mechanisms to break the deadlock.

To log more information about deadlocks, set the database manager configuration parameter *notifylevel* to four. The administration notification log stores information that includes the object, the lock mode, and the application holding the lock on the object. The current dynamic SQL statement or static package name might also be logged. The dynamic SQL statement is logged only if *notifylevel* is four.

Related concepts:

- "Locks and concurrency control" on page 59
- "Deadlocks between applications" on page 14

Related tasks:

- "Correcting lock escalation problems" on page 70

Related reference:

- "Diagnostic Error Capture Level configuration parameter - diaglevel" on page 507
- "Lock Timeout configuration parameter - locktimeout" on page 430

Guidelines for locking

Consider the following guidelines when you tune locking for concurrency and data integrity:

- Create small units of work with frequent COMMIT statements to promote concurrent access of data by many users.

  Include COMMIT statements when your application is logically consistent, 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.

- Specify an appropriate isolation level.

  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.

- Use the LOCK TABLE statement appropriately.

  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.

  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.

- Use ALTER TABLE statements in applications.

The ALTER TABLE statement with the LOCKSIZE parameter is an alternative to the LOCK TABLE statement. The LOCKSIZE parameter lets you specify a lock granularity of either ROW locks or TABLE locks for the next table access.

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 might be reduced because all locks are on the complete table. Neither choice prevents normal lock escalation.

- Close cursors to release the locks that they hold.

  When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, the database manager attempts to release all read locks that have been held for the cursor. Table read locks are IS, S, and U table locks. Row-read locks are S, NS, and U row locks. Block-read locks are IS, S, and U block locks.

  The WITH RELEASE clause has no effect on 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 are acquired.

  In CLI applications, the DB2® CLI connection attribute SQL_ATTR_CLOSE_BEHAVIOR can be used to achieve the same results as CLOSE CURSOR WITH RELEASE.
- In a partitioned database, when you changing the configuration parameters that affecting locking, ensure that the changes are made to all of the partitions.

Related concepts:

- “Locks and concurrency control” on page 59
- “Lock attributes” on page 61
- “Locks and performance” on page 63
- “Factors that affect locking” on page 83

Correcting lock escalation problems

The database manager can automatically escalate locks from row or block level to table level. The *maxlocks* database configuration parameter specifies when lock escalation is triggered. The table that acquires the lock that triggers lock escalation might not be affected. Locks are first escalated for the table with the most locks, beginning with tables for which long object (LOBs) and long VARCHAR descriptors are locked, then the table with the next highest

number of locks, and so on, until the number of locks held is decreased to about half of the value specified by *maxlocks*.

In a well designed database, lock escalation rarely occurs. If lock escalation reduces concurrency to an unacceptable level, however, you need to analyze the problem and decide how to solve it.

Prerequisites:

Ensure that lock escalation information is recorded. Set the database manager configuration parameter *notifylevel* to 3, which is the default, or to 4. At *notifylevel* of 2, only the error SQLCODE is reported. At *notifylevel* of 3 or 4, when lock escalation fails, information is recorded for the error SQLCODE and the table for which the escalation failed. The current SQL statement is logged only if it is a currently executing, dynamic SQL statement and *notifylevel*is set to 4.

Procedure:

Follow these general steps to diagnose the cause of unacceptable lock escalations and apply a remedy:

1. Analyze in the administration notification log on all tables for which locks are escalated. This log file includes the following information:
   - 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. Usually, an "S," or Share lock, or an "X," or eXclusive lock is acquired.
   - The internal return code of the result of the acquisition of the new table lock level.
2. Use the information in administration notification log to decide how to resolve the escalation problem. Consider the following possibilities:
   - Increase the number of locks allowed globally by increasing the value of the *maxlocks* or the *locklist* parameters, or both, in the database configuration file. In a partitioned database, make this change on all partitions.

     You might choose this method 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.

- Adjust the process or processes that caused the escalation. For these processes, you might issue LOCK TABLE statements explicitly.
- Change the degree of isolation. Note that this may lead to decreased concurrency, however.
- Increase the frequency of commits to reduce the number of locks held at a given time.
- Consider frequent COMMIT statements for transactions that require long VARCHAR or various kinds of long object (LOB) data. Although this kind of data is not retrieved from disk until the result set is materialized, the descriptor is locked when the data is first referenced. As a result, many more locks might be held than for rows that contain more ordinary kinds of data.

Related reference:

- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
- "Diagnostic Error Capture Level configuration parameter - diaglevel" on page 507

Lock type compatibility

The following table displays information about the circumstances in which a lock request can be granted when 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 a requestor is waiting for a lock. A **yes** indicates that the lock is granted unless an earlier requestor 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**	**X**	**Z**	**NW**	**W**
none	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IN	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
NS	yes	yes	yes	yes	yes	no	no	yes	no	no	yes	no
S	yes	yes	yes	yes	yes	no	no	yes	no	no	no	no
IX	yes	yes	yes	no	no	yes	no	no	no	no	no	no
SIX	yes	yes	yes	no	no	no	no	no	no	no	no	no
U	yes	yes	yes	yes	yes	no	no	no	no	no	no	no
X	yes	yes	no	no	no	no	no	no	no	no	no	no
Z	yes	no	no	no	no	no	no	no	no	no	no	no
NW	yes	yes	no	yes	no	no	no	no	no	no	no	yes
W	yes	yes	no	no	no	no	no	no	no	no	yes	no

Note:

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

Note:

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

Related concepts:

- “Locks and concurrency control” on page 59
- “Lock attributes” on page 61
- “Locks and performance” on page 63

Related reference:

- “Lock modes and access paths for standard tables” on page 74
- “Locking for block index scans for MDC tables” on page 80

Lock modes and access paths for standard tables

This topic includes reference information about locking methods for standard tables for different data-access plans.

The following tables list the types of locks obtained for standard tables at each level for different access plans. Each entry is made up of two parts: table lock and row lock. A dash indicates that a particular level of locking is not done.

Note: In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used.

Table 5. Lock Modes for Table Scans

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
Access Method: ***Table scan with no predicates***					
RR	S/-	U/-	SIX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X
Access Method: ***Table Scan with predicates***					
RR	S/-	U/-	SIX/X	U/-	SIX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Note: At UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks to an IS table lock and NS row locks.

Table 6. Lock Modes for RID Index Scans

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: ***RID index scan with no predicates***					
RR	S/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X

Table 6. Lock Modes for RID Index Scans (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
CS	IS / NS	IX / U	IX / X	IX / X	IX / X
UR	IN / -	IX / U	IX / X	IX / X	IX / X
Access Method: *RID index scan with a single qualifying row*					
RR	IS / S	IX / U	IX / X	IX / X	IX / X
RS	IS / NS	IX / U	IX / X	IX / X	IX / X
CS	IS / NS	IX / U	IX / X	IX / X	IX / X
UR	IN / -	IX / U	IX / X	IX / X	IX / X
Access Method: *Index scan with start and stop predicates only*					
RR	IS / S	IX / S	IX / X	IX / X	IX / X
RS	IS / NS	IX / U	IX / X	IX / X	IX / X
CS	IS / NS	IX / U	IX / X	IX / X	IX / X
UR	IN / -	IX / U	IX / X	IX / X	IX / X
Access Method: *Index Scan with index and other predicates (sargs, resids) only*					
RR	IS / S	IX / S	IX / X	IX / S	IX / X
RS	IS / NS	IX / U	IX / X	IX / U	IX / X
CS	IS / NS	IX / U	IX / X	IX / U	IX / X
UR	IN / -	IX / U	IX / X	IX / U	IX / X

The following table 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
- Sorted for efficient prefetching

Table 7. Lock modes for index scans used for deferred data page access

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: *RID index scan with no predicates*					
RR	IS / S	IX / S		X / -	
RS	IN / -	IN / -		IN / -	

Table 7. Lock modes for index scans used for deferred data page access (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
CS	IN / -	IN / -		IN / -	
UR	IN / -	IN / -		IN / -	
Access Method: ***Deferred Data Page Access, after a RID index scan with no predicates***					
RR	IN / -	IX / S	IX / X	X / -	X / -
RS	IS / NS	IX / U	IX / X	IX / X	IX / X
CS	IS / NS	IX / U	IX / X	IX / X	IX / X
UR	IN / -	IX / U	IX / X	IX / X	IX / X
Access Method: ***RID index scan with predicates (sargs, resids)***					
RR	IS / S	IX / S		IX / S	
RS	IN / -	IN / -		IN / -	
CS	IN / -	IN / -		IN / -	
UR	IN / -	IN / -		IN / -	
Access Method: ***RID 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 a RID index scan with start and stop predicates only***					
RR	IN / -	IX / S	IX / X	IX / X	IX / X
RS	IS / NS	IX / U	IX / X	IX / U	IX / X
CS	IS / NS	IX / U	IX / X	IX / U	IX / X
UR	IS / -	IX / U	IX / X	IX / U	IX / X
Access Method: ***Deferred data-page Access, after a RID index scan with predicates***					
RR	IN / -	IX / S	IX / X	IX / S	IX / X
RS	IS / NS	IX / U	IX / X	IX / U	IX / X
CS	IS / NS	IX / U	IX / X	IX / U	IX / X
UR	IN / -	IX / U	IX / X	IX / U	IX / X

Related concepts:

- "Lock attributes" on page 61
- "Locks and performance" on page 63

Related reference:

- "Lock type compatibility" on page 72
- "Lock modes for table and RID index scans of MDC tables" on page 77
- "Locking for block index scans for MDC tables" on page 80

Lock modes for table and RID index scans of MDC tables

In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used. The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not used.

Table 8. Lock Modes for Table Scans

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
Access Method: ***Table scan with no predicates***					
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
Access Method: ***Table Scan with predicates on dimension columns only***					
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
Access Method: ***Table Scan with other predicates (sargs, resids)***					
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

The following two tables show lock modes for RID indexes on MDC tables.

Table 9. Lock Modes for RID Index Scans

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: ***RID index scan with no predicates***					
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X
Access Method: ***RID index scan with single qualifying row***					
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X
Access Method: ***RID index scan with start and stop predicates only***					
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
Access Method: ***Index scan with index predicates only***					
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
Access Method: ***Index scan with other predicates (sargs, resids)***					
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Note: In the following table, which shows lock modes for RID index scans used for deferred data-page access, at UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks are upgraded

to an IS table lock, an IS block lock, and NS row locks.

Table 10. Lock modes for RID index scans used for deferred data-page access

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: ***RID index scan with no predicates***					
RR	IS / S / S	IX / IX / S		X / - / -	
RS	IN / IN / -	IN / IN / -		IN / IN / -	
CS	IN / IN / -	IN / IN / -		IN / IN / -	
UR	IN / IN / -	IN / IN / -		IN / IN / -	
Access Method: ***Deferred data-page access after a RID index scan with no predicates***					
RR	IN / IN / -	IX / IX / S	IX / IX / X	X / - / -	X / - / -
RS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / X	IX / IX / X
CS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / X	IX / IX / X
UR	IN / IN / -	IX / IX / U	IX / IX / X	IX / IX / X	IX / IX / X
Access Method: ***RID index scan with predicates (sargs, resids)***					
RR	IS / S / -	IX / IX / S		IX / IX / S	
RS	IN / IN / -	IN / IN / -		IN / IN / -	
CS	IN / IN / -	IN / IN / -		IN / IN / -	
UR	IN / IN / -	IN / IN / -		IN / IN / -	
Access Method: ***Deferred data-page access after a RID index scan with predicates (sargs, resids)***					
RR	IN / IN / -	IX / IX / S	IX / IX / X	IX / IX / S	IX / IX / X
RS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X
CS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X
UR	IN / IN / -	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X
Access Method: ***RID index scan with start and stop predicates only***					
RR	IS / IS / S	IX / IX / S		IX / IX / X	
RS	IN / IN / -	IN / IN / -		IN / IN / -	
CS	IN / IN / -	IN / IN / -		IN / IN / -	
UR	IN / IN / -	IN / IN / -		IN / IN / -	
Access Method: ***Deferred data-page access after a RID index scan with start and stop predicates only***					
RR	IN / IN / -	IX / IX / S	IX / IX / X	IX / IX / X	IX / IX / X

Table 10. Lock modes for RID index scans used for deferred data-page access (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Related concepts:

- "Locks and concurrency control" on page 59
- "Lock attributes" on page 61
- "Locks and performance" on page 63

Related reference:

- "Lock type compatibility" on page 72
- "Lock modes and access paths for standard tables" on page 74
- "Locking for block index scans for MDC tables" on page 80

Locking for block index scans for MDC tables

The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not done.

Table 11. Lock Modes for Index Scans

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: *With no predicates*					
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--
Access Method: *With dimension predicates only*					
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 11. Lock Modes for Index Scans (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
CS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / X / -	IX / X / -
UR	IN / IN / -	IX / IX / U	IX / IX / X	IX / X / -	IX / X / -
Access Method: ***With dimension start and stop predicates only***					
RR	IS / S / -	IX / IX / S	IX / IX / S	IX / IX / S	IX / IX / S
RS	IX / IX / S	IX / IX / U	IX / IX / X	IX / IX / -	IX / IX / -
CS	IX / IX / S	IX / IX / U	IX / IX / X	IX / IX / -	IX / IX / -
UR	IN / IN / -	IX / IX / U	IX / IX / X	IX / IX / -	IX / IX / -
Access Method: ***Index Scan with predicates***					
RR	IS / S / -	IX / IX / S	IX / IX / X	IX / IX / S	IX / IX / X
RS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X
CS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X
UR	IN / IN / -	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X

The following table lists lock modes for block index scans used for deferred data-page access:

Table 12. Lock modes for block index scans used for deferred data-page access

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: ***Block index scan with no predicates***					
RR	IS / S / --	IX / IX / S		X / -- / --	
RS	IN / IN / --	IN / IN / --		IN / IN / --	
CS	IN / IN / --	IN / IN / --		IN / IN / --	
UR	IN / IN / --	IN / IN / --		IN / IN / --	
Access Method: ***Deferred data-page access after a block index scan with no predicates***					
RR	IN / IN / --	IX / IX / S	IX / IX / X	X / -- / --	X / -- / --
RS	IS / IS / NS	IX / IX / U	IX / IX / X	X / X / --	X / X / --
CS	IS / IS / NS	IX / IX / U	IX / IX / X	X / X / --	X / X / --
UR	IN / IN / --	IX / IX / U	IX / IX / X	X / X / --	X / X / --

Table 12. Lock modes for block index scans used for deferred data-page access (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
Access Method: *Block index scan with dimension predicates only*					
RR	IS / S / --	IX / IX / --		IX / S / --	
RS	IS / IS / NS	IX / -- / --		IX / -- / --	
CS	IS / IS / NS	IX / -- / --		IX / -- / --	
UR	IN / IN / --	IX / -- / --		IX / -- / --	
Access Method: *Deferred data-page access after a block index scan with dimension predicates only*					
RR	IN / IN / --	IX / IX / S	IX / IX / X	IX / S / --	IX / X / --
RS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / U / --	IX / X / --
CS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / U / --	IX / X / --
UR	IN / IN / --	IX / IX / U	IX / IX / X	IX / U / --	IX / X / --
Access Method: *Block index scan with start and stop predicates only*					
RR	IS / S / --	IX / IX / --		IX / X / --	
RS	IN / IN / --	IN / IN / --		IN / IN / --	
CS	IN / IN / --	IN / IN / --		IN / IN / --	
UR	IN / IN / --	IN / IN / --		IN / IN / --	
Access Method: *Deferred data-page access after a block index scan with start and stop predicates only*					
RR	IN / IN / --	IX / IX / X		IX / X / --	
RS	IS / IS / NS	IN / IN / --		IN / IN / --	
CS	IS / IS / NS	IN / IN / --		IN / IN / --	
UR	IS / -- / --	IN / IN / --		IN / IN / --	
Access Method: *Block index scan other predicates (sargs, resids)*					
RR	IS / S / --	IX / IX / --		IX / IX / --	
RS	IN / IN / --	IN / IN / --		IN / IN / --	
CS	IN / IN / --	IN / IN / --		IN / IN / --	
UR	IN / IN / --	IN / IN / --		IN / IN / --	
Access Method: *Deferred data-page access after a block index scan with other predicates (sargs, resids*					
RR	IN / IN / --	IX / IX / S	IX / IX / X	IX / IX / S	IX / IX / X

Table 12. Lock modes for block index scans used for deferred data-page access (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X
CS	IS / IS / NS	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X
UR	IN / IN / --	IX / IX / U	IX / IX / X	IX / IX / U	IX / IX / X

Related concepts:

- "Locks and performance" on page 63

Related reference:

- "Lock type compatibility" on page 72
- "Lock modes and access paths for standard tables" on page 74
- "Lock modes for table and RID index scans of MDC tables" on page 77

Factors that affect locking

The following factors affect the mode and granularity of database manager locks:

- The type of processing that the application performs
- The data access method
- Whether indexes are type-2 or type-1
- Various configuration parameters

Related concepts:

- "Locks and concurrency control" on page 59
- "Lock attributes" on page 61
- "Locks and performance" on page 63
- "Guidelines for locking" on page 68
- "Index cleanup and maintenance" on page 302
- "Locks and types of application processing" on page 84
- "Locks and data-access methods" on page 85
- "Index types and next-key locking" on page 86

Factors that affect locking

Locks and types of application processing

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

- Read-only

  This type includes all select statements that are intrinsically read-only, have an explicit FOR READ ONLY clause, or are ambiguous but which the SQL compiler assumes to be read-only because of the value of the BLOCKING option that the PREP or BIND command specifies. This processing type requires only Share locks (S, NS, or IS).

- Intent to change

  This type includes all select statements with the FOR UPDATE clause, or for which the SQL compiler interprets an ambiguous statement to imply that change is intended. This type uses Share and Update locks (S, U, and X for rows; IX, U, X, and S for blocks; IX, U, and X for tables).

- Change

  This type includes UPDATE, INSERT, and DELETE, but not UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. This type 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 data in a target table, based on the result from a sub-select statement, does two types of processing. The rules for read-only processing determine the locks for the tables returned in the sub-select statement. The rules for change processing determine the locks for the target table.

Related concepts:

- “Locks and concurrency control” on page 59
- “Lock attributes” on page 61
- “Locks and performance” on page 63
- “Guidelines for locking” on page 68
- “Deadlocks between applications” on page 14
- “Locks and data-access methods” on page 85
- “Index types and next-key locking” on page 86

Related tasks:

- “Correcting lock escalation problems” on page 70

Related reference:

- "Lock type compatibility" on page 72

Locks and data-access methods

An *access plan* is the method that the optimizer selects to retrieve data from a specific table. The access plan can have a significant effect on lock modes. For example, when an index scan is used to locate a specific row, the optimizer will probably choose row-level locking (IS) for the table. For example, if the EMPLOYEE table that has an index on employee number (EMPNO), access through an index might be used to select information for a single employee with a statement that contains the following SELECT clause:

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

If an index is not used, the entire table must be scanned in sequence to find the selected rows, and may thus acquire a single table level lock (S). For example, if there is no index on the column SEX, a table scan might be used to select all male employees with a a statement that contains the following SELECT clause:

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

Note: Cursor controlled 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 is always obtained to perform the update or delete.

Reference tables provide detailed information about which locks are obtained for what kind of access plan.

Deferred access of the data pages implies that access to the row occurs in two steps, which results in more complex locking scenarios. The timing of lock aquisition and the persistence of the locks depend on the isolation level. Because the Repeatable Read isolation level retains all locks 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, locks are not acquired during the first step and rely on the reapplication of all predicates to ensure that only qualifying rows are returned.

Related concepts:

- "Locks and concurrency control" on page 59

- “Lock attributes” on page 61
- “Locks and performance” on page 63
- “Guidelines for locking” on page 68
- “Locks and types of application processing” on page 84
- “Index types and next-key locking” on page 86

Related tasks:

- “Correcting lock escalation problems” on page 70

Related reference:

- “Lock type compatibility” on page 72
- “Lock modes and access paths for standard tables” on page 74
- “Lock modes for table and RID index scans of MDC tables” on page 77
- “Locking for block index scans for MDC tables” on page 80

Index types and next-key locking

As transactions cause changes to type-1 indexes, some next-key locking occurs. For type-2 indexes, minimal next-key locking occurs.

- Next-key locking for type 2 indexes

  Next-key locking occurs when a key is inserted into an index.

  During insertion of a key into an index, the row that corresponds to the key that will follow the new key in the index is locked only if that row is currently locked by an RR index scan. The lock mode used for the next-key lock is NW. This next-key lock is released before the key insertion is actually performed. Key insertion occurs when a row is inserted into a table.

  When updates to a row result in a change to the value of the index key for that row, key insertion also occurs because the original key value is marked deleted and the new key value is inserted into the index. For updates that affect only the include columns of an index, the key can be updated in place and no next-key locking occurs.

  During RR scans, the row that corresponds to the key that follows the end of the scan range is locked in S mode. If no keys follow the end of the scan range, an end-of-table lock is acquired to lock the end of the index. If the key that follows the end of the scan range is marked deleted, the scan continues to lock the corresponding rows until it finds a key that is not marked deleted, when it locks the corresponding row for that key, or until the end of the index is locked.
- Next-key locking for type-1 indexes:

Next-key locks occur during deletes and inserts to indexes and during index scans. When a row is updated in, deleted from, or inserted into a table, an X lock is obtained on that row. For insertions this might be downgraded to a W lock.

When the key is deleted from the table index or inserted into it, the table row that corresponds to the key that follows the deleted or inserted key in the index is locked. For updates that affect the value of the key, the original key value is first deleted and the new value is inserted, so two next-key locks are acquired. The duration of these locks is determined as follows:

- During index key deletion, the lock mode on the next key is X and the lock is held until commit time.
- During index key insertion, the lock mode on the next key is NW. This lock is acquired only if there is contention for the lock, in which case the lock is released before the key is actually inserted into the index.
- During RR scans, the table row that corresponds to the key just beyond the end of the index scan range is locked in S mode and is held until commit time.
- During CS/RS scans, the row corresponding to the key just beyond the end of the index scan range is locked in NS mode if there is contention for the lock. This lock is released once the end of the scan range is verified.

The next-key locking for type-1 indexes during key insertions and key deletion might result in deadlocks. The following example shows how two transactions could deadlock. With type 2 indexes, such deadlocks do not occur.

Consider the following example of an index that contains 6 rows with the following values: 1 5 6 7 8 12.

1. Transaction 1 deletes the row with key value 8. The row with value 8 is locked in X mode. When the corresponding key from the index is deleted, the row with value 12 is locked in X mode.
2. Transaction 2 deletes the row with key value 5. The row with value 5 is locked in X mode. When the corresponding key from the index is deleted, the row with value 6 is locked in X mode.
3. Transaction 1 inserts a row with key value 4. This row is locked in W mode. When inserting the new key into the index is attempted, the row with value 6 is locked in NW mode. This lock attempt will wait on the X lock that transaction 2 has on this row.
4. Transaction 2 inserts a row with key value 9. This row is locked in W mode. When inserting the new key into the index is attempted, the row with key value 12 is locked in NW mode. This lock attempt will wait on the X lock that transaction 1 has on this row.

When type-1 indexes are used, this scenario will result in a deadlock and one of these transactions will be rolled back.

Related concepts:

- "Advantages and disadvantages of indexes" on page 294
- "Index performance tips" on page 299
- "Index structure" on page 31
- "Index reorganization" on page 303
- "Online index defragmentation" on page 305
- "Index cleanup and maintenance" on page 302

Optimization factors

This section describes the factors to consider when you specify the optimization class for queries.

Optimization class guidelines

When you compile an SQL query, you can specify an optimization class that determines how the optimizer chooses the most efficient access plan for that query. Although you can specify optimization techniques individually to improve runtime performance for the query, the more optimization techniques you specify, the more time and system resources query compilation will require.

Note: In a federated database query, the optimization class does not apply to the remote optimizer.

Setting the optimization class can provide some of the advantages of explicitly specifying optimization techniques, particularly for the following reasons:

- To manage very small databases or very simple dynamic queries
- To accommodate memory limitations at compile time on your database server
- To reduce the query compilation time, such as PREPARE.

Most statements can be adequately optimized with a reasonable amount of resources by using optimization class 5, which is the default query optimization class. At a given optimization class, the query compilation time and resource consumption 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.

Query optimization classes 1, 2, 3, 5, and 7 are all suitable for general-purpose use. Consider class 0 only if you require further reductions in query compilation time and you know that the SQL statements are extremely simple.

Tip: To analyze queries that run a long time, run the query with db2batch to find out how much time is spent in compilation and how much is spent in execution.If compilation requires more time, reduce the optimization class. If execution requires more time, consider a higher optimization class.

When you select an optimization class, consider the following general guidelines:

- Start by using the default query optimization class, class 5.
- To use a class other than the default, try class 1, 2 or 3 first. Classes 0, 1, and 2 use the Greedy join enumeration algorithm.
- 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 low optimization class (0 or 1) for queries having very short run-times of less than one second. Such queries tend to have the following characteristics:
  - Access to a single or only a few tables
  - Fetch a single or only a few rows
  - Use fully qualified, unique indexes.

  Online transaction processing (OLTP) transactions are good examples of this kind of SQL.
- Use a higher optimization class (3, 5, or 7) for longer running queries that take more than 30 seconds.
- Classes 3 and above use the Dynamic Programming join enumeration algorithm. This algorithm considers many more alternative plans, and might incur significantly more compilation time than classes 0, 1, and 2, especially as the number of tables increases.
- Use optimization class 9 only if you have specific extraordinary optimization requirements for a query.

Complex queries might require different amounts of optimization to select the best access plan. Consider using higher optimization classes for queries that have 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 for which at least the default query optimization class should be used.

Use higher query optimization classes for SQL that was produced by a query generator. Many query generators create inefficient SQL. Poorly written queries, including those produced by a query generator, require additional optimization to select a good access plan. Using query optimization class 2 and higher can improve such SQL queries.

Related concepts:

- "Configuration parameters that affect query optimization" on page 163
- "Benchmark testing" on page 355
- "Optimization strategies for intra-partition parallelism" on page 206
- "Optimization strategies for MDC tables" on page 209

Related tasks:

- "Setting the optimization class" on page 93

Related reference:

- "Optimization classes" on page 90

Optimization classes

You can specify one of the following optimizer classes when you compile an SQL query:

0 - This class directs the optimizer to use minimal optimization to generate an access plan. This optimization class has the following characteristics:
- Non-uniform distribution statistics are not considered by the optimizer.
- Only basic query rewrite rules are applied.
- Greedy join enumeration occurs.
- Only nested loop join and index scan access methods are enabled.
- List prefetch and index ANDing are not used in generated access methods.
- The star-join strategy is not considered.

This class should only be used in circumstances that require the the lowest possible query compilation overhead. Query optimization class 0 is appropriate for an application that consists entirely of very simple dynamic SQL statements that access well-indexed tables.

1 - This optimization class has the following characteristics:

- Non-uniform distribution statistics are not considered by the optimizer.
- Only a subset of the query rewrite rules are applied.
- Greedy join enumeration occurs.
- List prefetch and index ANDing are not used in generated access methods although index ANDing is still used when working with the semijoins used in star joins.

Optimization class 1 is 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 significantly higher than class 1, while keeping the compilation cost significantly lower than classes 3 and above for complex queries. This optimization class has the following characteristics:

- All available statistics, including both frequency and quantile non-uniform distribution statistics, are used.
- All query rewrite rules are applied, including routing queries to materialized query tables, except computationally intensive rules that are applicable only in very rare cases.
- Greedy join enumeration is used.
- A wide range of access methods are considered, including list prefetch and materialized query table routing.
- The star-join strategy is considered, if applicable.

Optimization class 2 is similar to class 5 except that it uses Greedy join enumeration instead of Dynamic Programming. This class has the most optimization of all 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. Class 2 is recommended for very complex queries in a decision support or online analytic processing (OLAP) environment. In such environments, specific queries are rarely repeated exactly, so that a query access plan is unlikely to remain in the cache until the next occurrence of the query.

3 - This class requests a moderate amount of optimization. This class comes closest to matching the query optimization characteristics of DB2 for MVS/ESA, OS/390, or z/OS. This optimization class has the following characteristics:

- Non-uniform distribution statistics, which track frequently occurring values, are used if available.
- Most query rewrite rules are applied, including subquery-to-join transformations.
- Dynamic programming join enumeration, as follows:
  – Limited use of composite inner tables
  – Limited use of Cartesian products for star schemas involving look-up tables
- 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. This class improves access plans for queries with four or more joins. However, the optimizer might fail to consider a better plan that might be chosen with the default optimization class.

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

- All available statistics are used, including both frequency and quantile distribution statistics.
- All of the query rewrite rules are applied, including the routing of queries to materialized query tables, except for those computationally intensive rules which are applicable only in very rare cases.
- Dynamic programming join enumeration, as follows:
  – Limited use of composite inner tables
  – Limited use of Cartesian products for star schemas involving look-up tables
- A wide range of access methods are considered, including list prefetch, index ANDing, and materialized query 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 depends on the machine size and the number of predicates.

When the query optimizer reduces the amount of query optimization, 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 is 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. You might use this class to find out whether more comprehensive optimization would generate a better access plan for very complex and very long-running queries that use large tables. Use Explain and performance measurements to verify that a better plan has actually been found.

Related concepts:

- "Optimization class guidelines" on page 88
- "Optimization strategies for intra-partition parallelism" on page 206
- "Remote SQL generation and global optimization in federated databases" on page 220
- "Optimization strategies for MDC tables" on page 209

Related tasks:

- "Setting the optimization class" on page 93

Setting the optimization class

When you specify an optimization level, consider whether a query uses static or dynamic SQL, and whether the same dynamic SQL is repeatedly executed. 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. Because dynamic statements are bound and executed at run time, consider whether the overhead of additional optimization for dynamic statements improves overall performance. However, if the same dynamic SQL statement is executed

repeatedly, the selected access plan is cached. Such statements can use the same optimization levels as static SQL statements.

If you think that a query that might benefit from additional optimization, but you are not sure, or you are concerned about compilation time and resource usage, you might perform some benchmark testing.

Procedure:

To specify a query optimization class, follow these steps:

1. Analyze the performance factors either informally or with formal tests as follows:
   - For **dynamic** SQL statements, tests should compare the average run time for the statement. Use the following formula to estimate an average run time:

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

     In this formula, the `number of iterations` represents the number of times that you expect that the SQL statement might be executed each time it is compiled.

     Note: After the initial compilation, dynamic SQL statements are recompiled when a change to the environment requires it. If the environment does not change after an SQL statement is cached, it does not need to be compiled again because subsequent PREPARE statements re-use the cached statement.
   - For **static** SQL statements, compare the statement run times.

     Although you might also be interested in the compile time of static SQL, the total compile and run time for the statement is difficult to assess 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.
2. Specify the optimization class as follows:
   - **Dynamic** SQL statements use the optimization class specified by the CURRENT QUERY OPTIMIZATION special register that you set with the SQL statement SET. 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 might include a SET statement in the application program.

If the CURRENT QUERY OPTIMIZATION register has not been set, dynamic statements are bound using the default query optimization class. The default value for both dynamic and static SQL is determined by value of the database configuration parameter *dft_queryopt*. Class 5 is the default value of this parameter. The default values for the bind option and the special register are also read from the *dft_queryopt* database configuration parameter.

- **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 is used for the static SQL statements. To change the optimization class for such static SQL statements, use the BIND command. If you do not specify the optimization class, DB2 uses the default optimization as specified by *dft_queryopt* database configuration parameter.

Related concepts:

- "Optimization class guidelines" on page 88

Related reference:

- "Optimization classes" on page 90

Tuning applications

This section provides guidelines for tuning the queries that applications execute.

Guidelines for restricting select statements

The optimizer assumes that an application must retrieve all of the rows specified by SELECT statement. This assumption is most appropriate in OLTP and batch environments. However, in "browse" applications, queries often define a large potential answer set but they retrieve only first few rows, usually only as many rows as are required to fill the screen.

To improve performance for such applications, you can modify the SELECT statement in the following ways:

- Use the FOR UPDATE clause to specify the columns that could be updated by a subsequent positioned UPDATE statement.
- Use the FOR READ/FETCH ONLY clause to make the returned columns read only.
- Use the OPTIMIZE FOR *n* ROWS clause to give priority to retrieving the first *n* rows in the full result set.

- Use the FETCH FIRST *n* ROWS ONLY clause to retrieve only a specified number of rows.
- Use the DECLARE CURSOR WITH HOLD statement to retrieve rows one at a time.

Note: Row blocking is affected if you use the FETCH FIRST *n* ROWS ONLY or the OPTIMIZE FOR *n* ROWS clause or if you declare your cursor as SCROLLing.

The following sections describe the performance advantages of each method.

FOR UPDATE Clause

The FOR UPDATE clause limits the result set by including only the columns that can be updated by a subsequent positioned UPDATE statement. If you specify the FOR UPDATE clause without column names, all columns that can be updated in the table or view are included. If you specify column names, each name must be unqualified and must identify a column of the table or view.

You cannot use FOR UPDATE clause in the following circumstances:

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

Use the DB2® CLI connection attribute SQL_ATTR_ACCESS_MODE in CLI applications for the same purposes.

FOR READ or FETCH ONLY Clause

The FOR READ ONLY clause or FOR FETCH ONLY clause ensures that read-only results are returned. Because the result table from a SELECT on a view defined as read-only is also read only, this clause is permitted but has no effect.

For result tables where updates and deletes are allowed, specifying FOR READ ONLY may improve the performance of FETCH operations if the database manager can retrieve blocks of data instead of exclusive locks. Do not use the FOR READ ONLY clause for queries that are used in positioned UPDATE or DELETE statements.

The DB2 CLI connection attribute SQL_ATTR_ACCESS_MODE can be used in CLI applications for the same purposes.

OPTIMIZE FOR n ROWS Clause

The OPTIMIZE FOR clause declares the intent to retrieve only a subset of the result or to give priority to retrieving only the first few rows. The optimizer can then prefer access plans that minimize the response time for retrieving the first few rows. In addition, the number of rows that are sent to the client as a single block are bounded by the value of "n" in the OPTIMIZE FOR clause. Thus the OPTIMIZE FOR clause affects both how the server retrieves the qualifying rows from the database by the server, and how it returns the qualifying rows 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. To avoid many random synchronous I/Os, the optimizer would probably choose to use the list prefetch access method, which requires sorting the row identifiers of all rows that qualify. This sort causes a delay before the first qualifying rows can be returned to the application. To prevent this delay, add the OPTIMIZE FOR clause to the statement as follows:

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

In this case, the optimizer probably chooses to use the SALARY index directly because only the twenty employees with the highest salaries are retrieved. Regardless of how many rows might be blocked, a block of rows is returned to the client every twenty rows.

With the OPTIMIZE FOR clause the optimizer favors access plans that avoid bulk operations or interrupt the flow of rows, such as sorts. You are most likely to influence an access path by using OPTIMIZE FOR 1 ROW. Using this clause might have the following effects:

- Join sequences with composite inner tables are less likely because they require a temporary table.
- The join method might change. A nested loop join is the most likely choice, because it has low overhead cost and is usually more efficient to retrieve a few rows.
- An index that matches the ORDER BY clause is more likely because no sort is required for the ORDER BY.
- List prefetch is less likely because this access method requires a sort.

- Sequential prefetch is less likely because of the understanding that only a small number of rows is required.
- In a join query, the table with the columns in the ORDER BY clause is likely to be picked as the outer table if an index on the outer table provides the ordering needed for the ORDER BY clause.

Although the OPTIMIZE FOR clause applies to all optimization levels, it works best for optimization class 3 and higher because classes below 3 use Greedy join enumeration method. This method sometimes results in access plans for multi-table joins that do not lend themselves to quick retrieval of the first few rows.

The OPTIMIZE FOR clause does not prevent you from retrieving all the qualifying rows. If you do retrieve all qualifying rows, the total elapsed time might be significantly greater than if the optimizer had optimized for the entire answer set.

If a packaged application uses the call level interface (DB2 CLI or ODBC), you can use the OPTIMIZEFORNROWS keyword in the `db2cli.ini` configuration file to have DB2 CLI automatically append an OPTIMIZE FOR clause to the end of each query statement.

When data is selected 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 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 that the least-cost plan is local execution, the OPTIMIZE FOR clause is applied locally. In this case, the DB2 optimizer prefers 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 limited 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 affects the communications buffer size. The two values are considered independent of each other for optimization purposes.

FETCH FIRST n ROWS ONLY Clause

The FETCH FIRST *n* ROWS ONLY clause sets the maximum number of rows that can be retrieved. Limiting the result table to the first several rows can improve performance. Only *n* rows are retrieved regardless of the number of rows that the result set might otherwise contain.

If you specify both the FETCH FIRST clause and the OPTIMIZE FOR clause, the lower of the two values affects the communications buffer size. For optimization purposes the two values are independent of each other.

DECLARE CURSOR WITH HOLD Statement

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

If the transaction is rolled back, all open cursors are closed and all locks are released and LOB locators are freed.

The DB2 CLI connection attribute SQL_ATTR_CURSOR_HOLD can be used in CLI applications to achieve the same results. If a packaged application that uses the call level interface (DB2 CLI or ODBC), use the CURSORHOLD keyword in the `db2cli.ini` configuration file to have DB2 CLI automatically assume the WITH HOLD clause for every declared cursor.

Related concepts:

- "Query tuning guidelines" on page 101
- "Efficient SELECT statements" on page 101

Specifying row blocking to reduce overhead

Row blocking reduces database manager overhead for cursors by retrieving a *block* of rows in a single operation.

Note: The block of rows that you specify is a number of pages in memory. It is not a multi-dimensional (MDC) table block, which is physically mapped to an extent on disk.

Row blocking levels are specified by the following arguments to the BIND or PREP commands:

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.

Prerequisites:

Two database manager configuration parameters must be set appropriately. Both values are set as a number of pages of memory. Note the values of these parameters for use in block-size calculations.

- The database manager configuration parameter *aslheapsz* specifies application support layer heap size for local applications.
- The database manager configuration parameter *rqrioblk* specifies the size of the communication buffer between remote applications and their database agents on the database server.

Procedure:

To specify row blocking:

1. Use the values of the *aslheapsz* and *rqrioblk* configuration parameters to estimate how many rows are returned for each block. In both formulas *orl* is the output row length in bytes.
   - Use the following formula for local applications:

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

     The number of bytes per page is 4 096.
   - Use the following formula for remote applications:

     ```
     Rows per block = rqrioblk / orl
     ```
2. To enable row blocking, specify an appropriate argument to the BLOCKING option in the PREP or BIND commands.

   If you do not specify a BLOCKING option, the default row blocking type is UNAMBIG. For the command line processor and call level interface, the default row blocking type is ALL.

Note: 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

Related reference:

- "Application Support Layer Heap Size configuration parameter - aslheapsz" on page 417
- "Client I/O Block Size configuration parameter - rqrioblk" on page 420

Query tuning guidelines

Follow the query-tuning guidelines to fine-tune the SQL statements in an application program. The guidelines are intended to help you minimize the use of system resources and the time required to return results from large tables and complex queries.

Note: The optimization class that the optimizer uses might eliminate the need for some fine tuning because the SQL compiler can rewrite the SQL code into more efficient forms.

Note that the optimizer choice of an access plan 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 find out what adjustments might improve the access plan.

Related concepts:

- "Guidelines for restricting select statements" on page 95
- "Efficient SELECT statements" on page 101
- "Compound SQL guidelines" on page 103
- "Character-conversion guidelines" on page 105
- "Guidelines for stored procedures" on page 106
- "Parallel processing for applications" on page 107
- "Guidelines for sort performance" on page 284

Related tasks:

- "Specifying row blocking to reduce overhead" on page 99

Efficient SELECT statements

Because SQL is a flexible high-level language, you can write several different SELECT statements to retrieve the same data. However, the performance might vary for the different forms of the statement as well as for the different classes of optimization.

Consider the following guidelines for SELECT statements:

- Specify only columns that you need. Although it is simpler to specify all columns with an asterisk (*), unnecessary processing and return of unneeded columns results.
- Use predicates that restrict the answer set to only those rows that you require
- When the number of rows you need is significantly less than the total number of rows that might be returned, specify the OPTIMIZE FOR clause.

This clause affects both the choice of access plans and the number of rows that are blocked in the communication buffer.

- When the number of rows to be retrieved is small, specify only the OPTIMIZE FOR *k* ROWS clause. You do not need the FETCH FIRST n ROWS ONLY clause. However, if *n* is large and you want the first *k* rows quickly with a possible delay for the subsequent *k* rows, specify both clauses. The size of the communication buffers is based on the lesser of *n* and *k*. The following example shows both clauses:

  ```
  SELECT EMPNAME, SALARY FROM EMPLOYEE
     ORDER BY SALARY DESC
     FETCH FIRST 100 ROWS ONLY
     OPTIMIZE FOR 20 ROWS
  ```

- To take advantage of row blocking, specify the FOR READ ONLY or FOR FETCH ONLY clause to improve performance. In addition, concurrency improves because exclusive locks are never held on the rows retrieved. Additional query rewrites can also occur. Specifying the FOR READ ONLY or FOR FETCH ONLY clause as well as the BLOCKING ALL BIND option can improve the performance of queries against nicknames in a federated system in a similar way.
- For cursors that will be updated, specify the FOR UPDATE OF clause to allow the database manager to choose more appropriate locking levels initially and avoid potential deadlocks.
- Avoid numeric data type conversions whenever possible. When comparing values, it might 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 might result.

  If possible, use the following data types:
  - Character instead of varying character for short columns
  - Integer instead of float or decimal
  - Datetime instead of character
  - Numeric instead of character
- To decrease the possibility that a sort operation will occur, omit clauses or operations such as DISTINCT or ORDER BY if such operations are not required.
- To check for existence of rows in a table, select a single row. Either open a cursor and fetch one row or perform a a single-row (SELECT INTO) selection. Remember to check for the SQLCODE -811 error if more than one row is found.

  Unless you know that the table is very small, do not use the following statement to check for a non-zero value:

  ```
  SELECT COUNT(*) FROM  TABLENAME
  ```

  For large tables, counting all the rows impacts performance.

- If update activity is low and 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 with host variables, looping a subset of the host variables might improve performance.

The following suggestions apply specifically to SELECT statements that access several tables.

- Use join predicates to join 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. Indexes 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 because the database manager cannot use some join techniques. As a result, the most efficient join method may not be chosen.
- In a partitioned database environment, if possible ensure that both of the tables joined are partitioned on the join column.

Related concepts:

- "Guidelines for restricting select statements" on page 95
- "Query tuning guidelines" on page 101

Compound SQL guidelines

To reduce database manager overhead, you can group several SQL statements into a single executable block. Because the SQL statements in the block are substatements that could be executed individually, this kind of code is called *compound SQL*. In addition to reducing database manager overhead, compound SQL reduces the number of requests that have to be transmitted across the network for remote clients.

There are two types of compound SQL:

- **Atomic**

  The application receives a response from the database manager when all substatements have completed successfully or when one substatement ends in an error. If one substatement ends in an error, the entire block is considered to have ended in an error. Any changes made to the database within the block are rolled back.

  Atomic compound SQL is not supported with DB2 Connect

- **Not Atomic**

  The application receives a response from the database manager when all substatements have completed. All substatements within a block are

executed regardless of whether or not the preceding substatement 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.

Compound SQL is supported in stored procedures, which are also known as DARI routines, and in the following application development processes:

- Embedded static SQL
- DB2 Call Level Interface
- JDBC

Dynamic Compound SQL Statements

Dynamic compound statements are compiled by DB2® as a single statement. This statement can be used effectively for short scripts that require little control flow logic but significant data flow. For larger constructs with nested complex control flow, consider using SQL procedures.

In a dynamic compound statement you can use the following elements in declarations:

- SQL variables in variable declarations of substatements
- Conditions in the substatements based on the SQLSTATE values of the condition declaration
- One or more SQL procedural statements

Dynamic compound statements can also use several flow logic statements, such as the FOR statement, the IF statement, the ITERATE statement, and the WHILE statement.

If an error occurs in a dynamic compound statement, all prior SQL statements are rolled back and the remaining SQL statements in the dynamic compound statement are not processed.

A dynamic compound statement can be embedded in a trigger, SQL function, or SQL method, or issued through dynamic SQL statements. This executable statement can be dynamically prepared. No privileges are required to invoke the statement but the authorization ID associated with the statement must have the necessary privileges to invoke the SQL statements in the compound statement.

Related concepts:

- “Query tuning guidelines” on page 101

Character-conversion guidelines

Data conversion might be required to map data between application and database code pages when your application and database do not use the same code page. Because mapping and data conversion require additional overhead application performance improves if the application and database use the same code page or the identity collating sequence.

Character conversion occurs in the following circumstances:

- When a client or application runs in a code page that is different from the code page of the database that it accesses.

  The conversion occurs on the database server machine that receives the data. If the database server receives the data, character conversion is from the application code page to the database code page. If the application machine receives the data, conversion is from the database code page to the application code page.
- When a client or application that imports or loads a file runs in a code page different from the file being imported or loaded.

Character conversion does not occur for the following objects:

- File names.
- Data targeted for or coming from a column for which the FOR BIT DATA attribute is assigned, or data that is used in an SQL operation whose result is FOR BIT or BLOB data.
- A DB2® product or platform for which no supported conversion function to or from EUC or UCS-2 is installed. Your application receives an SQLCODE -332 (SQLSTATE 57017) error in this case.

The conversion function and conversion tables or DBCS conversion APIs that the database manager uses when it converts multi-byte code pages depends on the operating system environment.

Note: Character string conversions between multi-byte code pages, such as DBCS with EUC, might increase or decrease length of a string. In addition, code points assigned to different characters in the PC DBCS, EUC, and UCS-2 code sets might produce different results when same characters are sorted.

Extended UNIX® Code (EUC) Code Page Support

Host variables that use graphic data in C or C++ applications require special considerations that include special precompiler, application performance, and application design issues.

Many characters in both the Japanese and Traditional Chinese EUC code pages require special methods of managing database and client application support for graphic data, which require double byte characters. Graphic data from these EUC code pages is stored and manipulated using the UCS-2 code set.

Related concepts:

- "Guidelines for analyzing where a federated query is evaluated" on page 218

Related reference:

- "Conversion tables for code pages 923 and 924" in the *Administration Guide: Planning*
- "Conversion table files for euro-enabled code pages" in the *Administration Guide: Planning*

Guidelines for stored procedures

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

Processing a single SQL statement for a remote database requires sending two transmissions: one request and one receive. Because an application contains many SQL statements it requires many transmissions to complete its work.

However, when a database client uses a stored procedure that encapsulates many SQL statements, it requires only two transmissions for the entire process.

Stored procedures usually run in processes separate from the database agents. This separation requires the stored procedure and agent processes to communicate through a router. However, a special kind of stored procedure that runs in the agent process might improve performance, although it carries significant risks of corrupting data and databases.

These risky stored procedures are those created as *not fenced*. For a not-fenced stored procedure, nothing separates the stored procedure from the database control structures that the database agent uses. If a DBA wants to ensure that the stored procedure operations will not accidentally or maliciously damage the database control structures, the *not fenced* option is omitted.

Because of the risk of damaging your database, use *not fenced* stored procedures **only** when you need the maximum possible performance benefits.

In addition, make absolutely sure 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 occurs while running a not-fenced stored procedure, the database manager determines whether the error occurred in the application or database manager code and performs the appropriate recovery.

A not-fenced stored procedure can corrupt the database manager beyond recovery, possibly resulting in lost data and the possibility of a corrupt database. Exercise extreme caution when you run not-fenced trusted stored procedures. In almost all cases, the proper performance analysis of an application results in the good performance without using not-fenced stored procedures. For example, triggers might improve performance.

Related concepts:

- "Query tuning guidelines" on page 101

Parallel processing for applications

DB2® supports parallel environments primarily on symmetric multi-processor (SMP) machines, but also to a limited extent on uniprocessor machines. In SMP machines, more than one processor can access the database, allowing parallel execution of complex SQL requests to be divided among the processors.

To specify the degree of parallelism to implement when you compile an application, use the CURRENT DEGREE special register, or the DEGREE bind option. *Degree* refers to the number of parts of a query that execute concurrently. There is no strict relation between the number of processors and the value that you select for the degree of parallelism. You can specify more or less than the number of processors on the machine. Even for uniprocessor machines you can set a degree higher than one to improve performance in some ways. Note, however, that each degree of parallelism adds to the system memory and CPU overhead.

Some configuration parameters must be modified to optimize performance when you use parallel execution of queries. In particular, for an environment with a high degree of parallelism, you should review and modify configuration parameters that control the amount of shared memory and prefetching.

The following three configuration parameters control and manage intra-partition parallelism.

- The *intra_parallel* database manager configuration parameter enables or disables parallelism support.

- The *max_querydegree* database configuration parameter sets an upper limit for the degree of parallelism for any query in the database. This value overrides the CURRENT DEGREE special register and the DEGREE bind option.
- The *dft_degree* database configuration parameter sets the default value for the CURRENT DEGREE special register and the DEGREE bind option.

If a query is compiled 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 and the amount of activity on the system. Parallelism may be lowered before query execution if the system is heavily utilized. This occurs because 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.

To display information about the degree of parallelism chosen by the optimizer, use the SQL Explain Facility to display the access plan. Use the database system monitor to display information about the degree of parallelism actually used at runtime.

Parallelism in non-SMP environments

You can specify a degree of parallelism without having an SMP machine. For example, I/O-bound queries on a uniprocessor machine may benefit from declaring a degree of 2 or more. In this case, the processor might not have to wait for input or output tasks to complete before starting to process a new query. Declaring a degree of 2 or more does not directly control I/O parallelism on a uniprocessor machine, however. Utilities such as Load can control I/O parallelism independently of such a declaration. The keyword `ANY` can also be used to set the *dft_degree* database manager configuration parameter. The `ANY` keyword allows the optimizer to determine the degree of intra-partition parallelism.

Related concepts:

- "Explain tools" on page 228
- "Optimization strategies for intra-partition parallelism" on page 206

Related reference:

- "Maximum Query Degree of Parallelism configuration parameter - max_querydegree" on page 505
- "Enable Intra-Partition Parallelism configuration parameter - intra_parallel" on page 506

- “Default Degree configuration parameter - dft_degree” on page 491

Chapter 4. Environmental considerations

In addition to the factors that you consider when you design and code your application, which are described in Chapter 3, “Application considerations” on page 51, certain environmental factors can influence the access plan chosen for your application.

For more information about factors that affect the SQL optimizer directly, see Chapter 5, “System catalog statistics” on page 117.

When you tune applications and make changes to the environment, rebind your applications to ensure that the best access plan is used.

Database partition group impact on query optimization

In partitioned databases, the optimizer recognizes collocation of tables and uses this collocation when it determines the best access plan for a query. If tables are frequently involved in join queries, they should be divided among partitions in a partitioned database so that the rows from each table being joined are located on the same database partition. During the join operation, the collocation of the data from both joined tables prevents moving data from one partition to another. Place both tables in the same database partition group to ensure that the data from the tables is collocated.

In a partitioned database, depending on the size of the table, spreading data over more partitions reduces the estimated time (or cost) to execute a query. The number of tables, the size of the tables, the location of the data in those tables, and the type of query, such as whether a join is required, all affect the cost of the query.

Related concepts:

- “Join strategies in partitioned databases” on page 196
- “Join methods in partitioned databases” on page 198
- “Partitions in a partitioned database” on page 338

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 during query execution. When it selects 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. Using the rotational-latency formula,this would produce:

  ```
  (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 are arrays of disks (such as RAID), then you must take additional considerations into account when you attempt 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 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 mentioned above where you 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 provided through these two values 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 also considers 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 wait time associated with reading the data into the buffer pool.

  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 you create a table space and or use 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. Review the recommendations for sizing this parameter and make changes as needed to improve the data movement.

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, consider rebinding your applications and executing the RUNSTATS utility to collect the latest statistics about the indexes to ensure that the best access plans are used.

Related concepts:

- "Catalog statistics tables" on page 124
- "The SQL compiler process" on page 159

- "Illustration of the DMS table-space address map" on page 22

Server options affecting federated databases

A federated system is composed of a DB2® DBMS (the federated database) and one or more data sources. You identify the data sources to the federated database when you issue CREATE SERVER statements. When you issue these statements, you can include server options that refine and control aspects of federated system operations involving DB2 and the specified data source. To change server options later, use 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.

The server option values that you specify affect query pushdown analysis, global optimization and other aspects of federated database operations. For example, in the CREATE SERVER statement, you can specify performance statistics as server option values, such as the *cpu_ratio* option, which specifies the relative speeds of the CPUs at the data source and the federated server. You might also set the *io_ratio* option to a value that indicates the relative rates of the data I/O divides at the source and the federated server. When you execute the CREATE SERVER statement, 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), use the ALTER SERVER statement to update SYSCAT.SERVEROPTIONS with this change. The optimizer then uses the new information the next time it chooses an access plan for the data source.

Related reference:

- "ALTER SERVER statement" in the *SQL Reference, Volume 2*
- "SYSCAT.SERVEROPTIONS catalog view" in the *SQL Reference, Volume 1*
- "Federated Database System Support configuration parameter - federated" on page 520
- "CREATE SEQUENCE statement" in the *SQL Reference, Volume 2*

Chapter 5. System catalog statistics

Statistical data stored in the system catalogs helps the optimizer choose the best access plan for queries. Make sure that you execute RUNSTATS to update this statistical data:

- At frequent regular intervals for tables whose contents changes continually.
- After each operation that adds or changes data in a significant number of table rows. Such operations include batch updates and data loading that adds rows.

Catalog statistics

When the SQL compiler optimizes SQL query plans, its decisions are heavily influenced by statistical information about the size of the database tables and indexes. The optimizer also uses information about the distribution of data in specific columns of tables and indexes if these columns are used to select rows or join tables. The optimizer uses this information to estimate the costs of alternative access plans for each query. When significant numbers of table rows are added or removed, or if data in columns for which you collect statistics is updated, execute RUNSTATS again to update the statistics.

Statistical information is collected for specific tables and indexes in the local database when you execute the RUNSTATS utility. Statistics are collected only for the table partition that resides on the partition where you execute the utility or the first partition in the database partition group that contains the table. The collected statistics are stored in the system catalog tables.

Note: Because the RUNSTATS utility does not support use of nicknames, you update statistics differently for federated database queries. If queries access a federated database, execute RUNSTATS for the tables in all databases, then drop and recreate the nicknames that access remote tables to make the new statistics available to the optimizer.

In addition to table size and data distribution information, you can also collect statistical information about the cluster ratio of indexes, the number of leaf pages in indexes, the number of table rows that overflow their original pages, and the number of filled and empty pages in a table. You use this information to decide when to reorganize tables and indexes.

Consider these tips to improve the efficiency of RUNSTATS and the usefulness of the collected statistics:

- Collect statistics only for the columns used to join tables or in the WHERE, GROUP BY, and similar clauses of queries. If these columns are indexed, you can specify the columns with the ONLY ON KEY COLUMNS clause for the RUNSTATS command.
- Customize the values for *num_freqvalues* and *num_quantiles* for specific tables and specific columns in tables.
- Collect DETAILED index statistics with the SAMPLE DETAILED clause to reduce the amount of background calculation performed for detailed index statistics. The SAMPLE DETAILED clause reduces the time required to collect statistics, and produces adequate precision in most cases.
- When you create an index for a populated table, add the COLLECT STATISTICS clause to create statistics as the index is created.

Distribution statistics are not collected:
- When the *num_freqvalues* and *num_quantiles* configuration parameters are set to zero (0)
- When the distribution of data is known, such as when each data value is unique.
- When the column is a data type for which statistics are never collected. These data type are LONG, large object (LOB), or structured columns.
- For row types in sub-tables, the table level statistics NPAGES, FPAGES, and OVERFLOW are not collected.
- If quantile distributions are requested, but there is only one non-NULL value in the column
- For extended indexes or declared temporary tables

Note: You can perform a RUNSTATS on a declared temporary table, but the resulting statistics are not stored in the system catalogs because declared temporary tables do not have catalog entries. However, the statistics are stored in memory structures that represent the catalog information for declared temporary tables. In some cases, therefore, it might be useful to perform a RUNSTATS on these tables.

Related concepts:
- “Catalog statistics tables” on page 124
- “Statistical information that is collected” on page 131
- “Catalog statistics for modeling and what-if planning” on page 147
- “Statistics for modeling production databases” on page 149
- “General rules for updating catalog statistics manually” on page 152

Related tasks:
- “Collecting catalog statistics” on page 120

Related reference:

- "Number of Frequent Values Retained configuration parameter - num_freqvalues" on page 493
- "Number of Quantiles for Columns configuration parameter - num_quantiles" on page 494
- "RUNSTATS Command" in the *Command Reference*

Collecting and analyzing catalog statistics

This section provides guidelines and instructions for collecting catalog statistics, as well as some hints for analyzing the collected data for better understanding of the data distribution, clustering, and so on.

Guidelines for collecting and updating statistics

The RUNSTATS command collects statistics on both the table and the index data to provide the optimizer with accurate information for access plan selection.

Note: By default, statistics are collected on a single node in a partitioned database, and the result is extrapolated for other nodes.

Use the RUNSTATS utility to collect statistics in the following situations:
- When data has been loaded into a table and the appropriate indexes have been created.
- When you create a new index on a table. You need execute RUNSTATS for only the new index if the table has not been modified since you last ran RUNSTATS on it.
- When a table has been reorganized with the REORG utility.
- When the table and its indexes have been extensively updated, by data modifications, deletions, and insertions. ("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 you want to compare current and previous statistics. If you update statistics at regular intervals you can discover performance problems early.
- When the prefetch quantity is changed.
- When you have used the REDISTRIBUTE DATABASE PARTITION GROUP utility.

  Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

To improve RUNSTATS performance and save disk space used to store statistics, consider specifying only the columns for which data distribution statistics should be collected.

Ideally, you should rebind application programs after running statistics. The query optimizer might choose a different access plan if it has new statistics.

If you do not have enough time to collect all of the statistics at one time, you might run RUNSTATS to update statistics on only a few tables and indexes at a time, rotating through the set of tables. If inconsistencies occur 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 during query optimization. For example, you first use RUNSTATS to gather table distribution statistics. Subsequently, you use RUNSTATS to gather index statistics. If inconsistencies occur as a result of activity on the table and are detected during query optimization, the warning message is issued. When this happens, you should run RUNSTATS again to update distribution statistics.

To ensure that the index statistics are synchronized with the table, execute RUNSTATS to collect both table and index statistics at the same time. 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 on all nodes.

Related concepts:

- "Catalog statistics" on page 117
- "Optimizer use of distribution statistics" on page 136

Related tasks:

- "Collecting catalog statistics" on page 120
- "Collecting distribution statistics for specific columns" on page 121
- "Collecting index statistics" on page 123

Collecting catalog statistics

You collect catalog statistics on tables and indexes to provide information that the optimizer uses to choose the best access plans for queries.

Prerequisites:

You must connect to the database that contains the tables and indexes and have one of the following authorization levels:

- sysadm
- sysctrl
- sysmaint

- dbadm
- CONTROL privilege on the table

Procedure:

To collect catalog statistics:

1. Connect to the database that contains the tables and indexes for which you want to collect statistical information.
2. From the DB2 command line, execute the RUNSTATS command with appropriate options. These options allow you to tailor the statistics that are collected for the queries that run against the tables and indexes.

   Note: By default, RUNSTATS collects statistics only for tables on the partition from which you execute it. If the database partition from which you execute RUNSTATS does not contain a table partition, the request is sent to the first database partition in the database partition group that holds a partition for the table and extrapolates the result to other partitions.

   For a complete list of RUNSTATS options, refer to the Command Reference.
3. When RUNSTATS is complete, issue a COMMIT statement to release locks.
4. Rebind packages that access tables and indexes for which you have regenerated statistical information.

To use a graphical user interface to specify options and collect statistics, use the Control Center.

Related concepts:

- "Catalog statistics" on page 117
- "Guidelines for collecting and updating statistics" on page 119

Related tasks:

- "Collecting distribution statistics for specific columns" on page 121
- "Collecting index statistics" on page 123
- "Determining when to reorganize tables" on page 288

Related reference:

- "RUNSTATS Command" in the *Command Reference*

Collecting distribution statistics for specific columns

For efficiency both of RUNSTATS and subsequent query-plan analysis, you might collect distribution statistics on only the table columns that queries use

in WHERE, GROUP BY, and similar clauses. You might also collect cardinality statistics on combined groups of columns. The optimizer uses such information to detect column correlation when it estimates selectivity for queries that reference the columns in the group.

In the following steps, the database is assumed to be **sales** and to contain the table **customers**, with indexes **custidx1** and **custidx2**.

Prerequisites:

You must connect to the database that contains the tables and indexes and have one of the following authorization levels:

- sysadm
- sysctrl
- sysmaint
- dbadm
- CONTROL privilege on the table

Note: By default, RUNSTATS collects statistics only for tables on the partition from which you execute it. If the database partition from which you execute RUNSTATS does not contain a table partition, the request is sent to the first database partition in the database partition group that holds a partition for the table.

You should collect statistics for all partitions if you suspect that data distribution is skewed on any partition or if you want to ensure that statistical information is represents the data distribution more accurately.

Procedure:

To collect statistics on specific columns:

1. Connect to the **sales** database.
2. Execute one of the following commands at the DB2 command line, depending on your requirements:
   - To collect distribution statistics on columns **zip** and **ytdtotal**:

     ```
     RUNSTATS ON TABLE sales.customers
          WITH DISTRIBUTION ON COLUMNS (zip, ytdtotal)
     ```

   - To collect distribution statistics on the same columns, but adjust the distribution defaults:

     ```
     RUNSTATS ON TABLE sales.customers
          WITH DISTRIBUTION ON
          COLUMNS (zip, ytdtotal NUM_FREQVALUES 50 NUM_QUANTILES 75)
     ```

- To collect distribution statistics on the columns indexed in **custidx1** and **custidx2**:

```
RUNSTATS ON TABLE sales.customer
        ON KEY COLUMNS
```

- To collect column statistics on the table only for specific columns **zip** and **ytdtotal** and a column group that includes **region** and **territory**:

```
RUNSTATS ON TABLE sales.customers
         ON COLUMNS (zip, (region, territory), ytdtotal)
```

You can also use the Control Center to collect distribution statistics.

Related concepts:

- "Catalog statistics tables" on page 124

Related tasks:

- "Collecting catalog statistics" on page 120
- "Collecting index statistics" on page 123

Collecting index statistics

Collect index statistics to allow the optimizer to evaluate whether an index should be used to resolve a query.

In the following steps, the database is assumed be **sales** and to contain the table **customers**, with indexes **custidx1** and **custidx2**.

Prerequisites:

You must connect to the database that contains the tables and indexes and have one of the following authorization levels:

- sysadm
- sysctrl
- sysmaint
- dbadm
- CONTROL privilege on the table

Executing RUNSTATS with the SAMPLED DETAILED option requires 2MB of the statistics heap. Allocate an additional 488 4K pages to the *stat_heap_sz* database configuration parameter setting for this additional memory requirement. If the heap appears to be too small, RUNSTATS returns an error before it attempts to collect statistics.

Procedure:

To collect detailed statistics for an index:

1. Connect to the **sales** database.
2. Execute one of the following commands at the DB2 command line, depending on your requirements:
   - To create detailed statistics on both **custidx1** and **custidx2**:

     ```
     RUNSTATS ON TABLE sales.customers AND DETAILED INDEXES ALL
     ```

   - To create detailed statistics on both indexes, but use sampling instead of performing detailed calculations for each index entry:

     ```
     RUNSTATS ON TABLE sales.customers AND SAMPLED DETAILED INDEXES ALL
     ```

   - To create detailed sampled statistics on indexes as well as distribution statistics for the table so that index and table statistics are consistent:

     ```
     RUNSTATS ON TABLE sales.customers
               WITH DISTRIBUTION ON KEY COLUMNS
               AND SAMPLED DETAILED INDEXES ALL
     ```

You can also use the Control Center to collect index and table statistics.

Related concepts:

- "Catalog statistics" on page 117
- "Catalog statistics tables" on page 124
- "Statistical information that is collected" on page 131
- "Detailed index statistics" on page 142

Related tasks:

- "Collecting catalog statistics" on page 120

Statistics collected

This section lists the catalog statistics tables and describes the use of the fields in these tables. Sections that follow the statistics tables descriptions explain the kind of data that can be collected and stored in the tables.

Catalog statistics tables

The following tables provide information about the system catalog tables that contain catalog statistics and the RUNSTATS options that collect specific statistics.

Table 13. Table Statistics (SYSCAT.TABLES and SYSSTAT.TABLES)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
FPAGES	number of pages being used by a table.	Yes	Yes

Table 13. Table Statistics (SYSCAT.TABLES and SYSSTAT.TABLES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
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 1)
ACTIVE_BLOCKS	for MDC tables, the total number of occupied blocks	Yes	No

Note:
1. If the table has no indexes defined and you request statistics for indexes, no new CARD statistics are updated. The previous CARD statistics are retained.

Table 14. Column Statistics (SYSCAT.COLUMNS and SYSSTAT.COLUMNS)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
COLCARD	column cardinality	Yes	Yes (Note 1)
AVGCOLLEN	average length of column	Yes	Yes (Note 1)
HIGH2KEY	second highest value in column	Yes	Yes (Note 1)
LOW2KEY	second lowest value in column	Yes	Yes (Note 1)
NUMNULLS	the number of NULLs in a column	Yes	Yes (Note 1)
SUB_COUNT	the average number of subelements	Yes	No (Note 2)
SUB_DELIM_LENGTH	average length of each delimiter separating each subelement	Yes	No (Note 2)

Note:
1. Column statistics are gathered for the first column in the index key.
2. These statistics provide information about data in columns that contain a series of subfields or subelements that are delimited by blanks. The SUB_COUNT and SUB_DELIM_LENGTH statistics are collected only for single-byte character set string columns of type CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC.

Table 15. Multicolumn Statistics (SYSCAT.COLGROUPS and SYSSTAT.COLGROUPS)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
COLGROUPCARD	cardinality of the column group	Yes	No

Note: The multicolumn distribution statistics listed in the following two tables are not collected by RUNSTATS. You can update them manually, however.

Table 16. Multicolumn Distribution Statistics (SYSCAT.COLGROUPDIST and SYSSTAT.COLGROUPDIST)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
TYPE	F = frequency value Q = quantile value	Yes	No
ORDINAL	Ordinal number of the column in the group	Yes	No
SEQNO	Sequence number *n* that represents the *n*th TYPE value	Yes	No
COLVALUE	the data value as a character literal or a null value	Yes	No

Table 17. Multicolumn Distribution Statistics 2 (SYSCAT.COLGROUPDISTCOUNTS and SYSSTAT.COLGROUPDISTCOUNTS)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
TYPE	F = frequency value Q = quantile value	Yes	No
SEQNO	Sequence number *n* that represents the *n*th TYPE value	Yes	No

Table 17. Multicolumn Distribution Statistics 2 (SYSCAT.COLGROUPDISTCOUNTS and SYSSTAT.COLGROUPDISTCOUNTS) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
VALCOUNT	If TYPE = F, VALCOUNT is the number of occurrences of COLVALUEs for the column group identified by this SEQNO. If TYPE = Q, VALCOUNT is the number of rows whose value is less than or equal to COLVALUEs for the column group with this SEQNO.	Yes	No
DISTCOUNT	If TYPE = Q, this column contains the number of distinct values that are less than or equal to COLVALUEs for the column group with this SEQNO. Null if unavailable.	Yes	No

Table 18. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
NLEAF	number of index leaf pages	No	Yes
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 18. 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 3)	No	Yes
FIRSTKEYCARD	number of distinct values in first column of the index	No	Yes
FIRST2KEYCARD	number of distinct values in first two columns of the index	No	Yes
FIRST3KEYCARD	number of distinct values in first three columns of the index	No	Yes
FIRST4KEYCARD	number of distinct values in first four columns of the index	No	Yes
FULLKEYCARD	number of distinct values in all columns of the index, excluding any key value in a type-2 index for which all RIDs are marked deleted	No	Yes
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 18. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
AVERAGE_SEQUENCE_PAGES	average number of index pages accessible in sequence. This is the number of index pages that the prefetchers can detect as being in sequence.	No	Yes
AVERAGE_RANDOM_PAGES	average number of random index pages between sequential page accesses	No	Yes
AVERAGE_SEQUENCE_GAP	gap between sequences	No	Yes
AVERAGE_SEQUENCE_FETCH_PAGES	average number of table pages accessible in sequence. This is the number of table pages that the prefetchers can detect as being in sequence when they fetch table rows using the index.	No	Yes
AVERAGE_RANDOM_FETCH_PAGES	average number of random table pages between sequential page accesses when fetching table rows using the index.	No	Yes
AVERAGE_SEQUENCE_FETCH_GAP	gap between sequences when fetching table rows using the index.	No	Yes

Table 18. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
NUMRIDS	the number of record identifiers (RIDs) in the index, including deleted RIDs in type-2 indexes.	No	Yes
NUMRIDS_DELETED	the total number of RIDs marked deleted in the index, except RIDs on leaf pages on which all record identifiers are marked deleted	No	Yes
NUM_EMPTY_LEAFS	the total number of leaf pages on which all record identifiers are marked deleted	No	Yes

Note:

1. Detailed index statistics are gathered by specifying the DETAILED clause on the RUNSTATS command.
2. CLUSTERFACTOR 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. 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 19. 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

Table 19. Column Distribution Statistics (SYSCAT.COLDIST and SYSSTAT.COLDIST) (continued)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
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

Note:

1. Column distribution statistics are gathered by specifying the WITH DISTRIBUTION clause on the RUNSTATS command. 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.

Related concepts:

- "Catalog statistics" on page 117
- "Statistical information that is collected" on page 131
- "Statistics for user-defined functions" on page 146
- "Statistics for modeling production databases" on page 149

Statistical information that is collected

When you execute the RUNSTATS utility for a table or for a table and its associated indexes, the following kinds of statistical information are always stored in the system catalog tables:

For a table and index:

- The number of pages in use
- The number of pages that contain rows
- The number of rows that overflow
- The number of rows in the table (cardinality)

- For MDC tables, the number of blocks that contain data

For each column in the table and the first column in the index key:
- The cardinality of the column
- The average length of the column
- The second highest value in the columns
- The second lowest value in the column
- The number of NULLs in the column

For groups of columns that you specify:
- A time-stamp based name for the column group
- The cardinality of the column group

For indexes only:
- The number of leaf pages
- The number of index levels
- The degree of clustering of the table data to this index
- The ratio of leaf pages on disk in index key order to the number of pages in the range of pages occupied by the index
- The number of distinct values in the first column of the index
- The number of distinct values in the first two, three, and four columns of the index
- The number of distinct values in all columns of the index
- The number of leaf pages located on disk in index key order, with few or no large gaps between them
- The number of pages on which all RIDs are marked deleted
- The number of RIDs marked deleted on pages on which not all RIDs are marked deleted

If you request detailed statistics for an index, you also store finer information about the degree of clustering of the table to the index and the page fetch estimates for different buffer sizes.

You can also collect the following kinds statistics about tables and indexes:
- Data distribution statistics

  The optimizer uses data distribution statistics to estimate efficient access plans for tables in which data is not evenly distributed and columns have a significant number of duplicate values.
- Detailed index statistics

  The optimizer uses detailed index statistics to determine how efficient it is to access a table through an index.
- Sub-element statistics

  The optimizer uses sub-element statistics for LIKE predicates, especially those that search for a pattern embedded within a string, such as `LIKE %disk%`.

Related concepts:

- "Catalog statistics" on page 117
- "Catalog statistics tables" on page 124

Related tasks:

- "Collecting catalog statistics" on page 120
- "Collecting distribution statistics for specific columns" on page 121
- "Collecting index statistics" on page 123

Distribution statistics

You can collect two kinds of data distribution statistics:

- Frequency statistics

  These statistics provide information about the column and the data value with the highest number of duplicates, the next highest number of duplicate values, and so on to the level specified by the value of the *num_freqvalues* database configuration parameter. To disable collection of frequent-value statistics, set *num_freqvalues* to 0.

  You can also set *num_freqvalues* as RUNSTATS options for each table and for specific columns.
- Quantile statistics

  These statistics provide information about how data values are distributed in relation to other values. Called K-quantiles, these statistics represent the value V at or below which at least K values lie. You can compute a K-quantile by sorting the values in ascending order. The K-quantile value is the value in the Kth position from the low end of the range.

  To specify the number of sections into which the column data values should be grouped, set the *num_quantiles* database configuration parameter to a value between 2 and 32,767. The default value is 20, which ensures an optimizer estimation error of a maximum of plus or minus 2.5% for any equality or less-than or greater-than predicate and a maximum error of plus or minus 5% for any BETWEEN predicate. To disable collection of quantile statistics, set num_quantiles to 0 or 1.

  You can set *num_quantiles* for each table and for specific columns.

Note: If you specify larger *num_freqvalues* and *num_quantiles* values, more CPU resources and memory, as specified by the *stat_heap_sz* database configuration parameter, are required when you execute RUNSTATS.

When to collect distribution statistics

To decide whether distribution statistics should be created and updated for a given table, consider the following two factors:

- Whether applications use 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.

- Whether data in columns is distributed uniformly.

  Create distribution statistics 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;
```

Two types of non-uniform data distributions might occur, possibly together:

- Data might be clustered in one or more sub-intervals instead of being evenly spread out between the highest and lowest data value. Consider the following column, in which 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

  Quantile statistics help the optimizer deal with this kind of data distribution.

  To help determine whether column data is not uniformly distributed, execute a query such as the following example:

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

- Duplicate data values might occur often. Consider a column in which data is distributed with the following frequencies:

Data Value	**Frequency**
20	5
30	10
40	10

Data Value	Frequency
50	25
60	25
70	20
80	5

To help the optimizer deal with duplicate values, create both quantile and frequent-value statistics.

When to collect index statistics only

You might collect statistics based only on index data in the following situations:

- A new index has been created since the RUNSTATS utility was run and you do not want to collect statistics again on the table data.
- There have been many changes to the data that affect the first column of an index.

What level of statistical precision to specify

To determine the precision with which distribution statistics are stored, you specify the database configuration parameters, *num_quantiles* and *num_freqvalues*. You can also specify these parameters as RUNSTATS options when you collect statistics for a table or for columns. The higher you set these values, the greater precision RUNSTATS uses when it create and updates distribution statistics. However, greater precision requires greater use of resources, both during RUNSTATS execution and in the storage required in the catalog tables.

For most databases, specify between 10 and 100 for the *num_freqvalues* database configuration parameter. Ideally, frequent-value statistics should be created such 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. The database manager might collect less than this number, because these statistics will only be collected for data values that occur more than once. If you need to collect only quantile statistics, set *num_freqvalues* to zero.

To set the number of quantiles, specify between 20 and 50 as the setting of the *num_quantiles* database configuration parameter. 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, specify at least 10 quantiles. More than 50 quantiles should be necessary only for extremely non-uniform data. If you need only frequent value statistics, set *num_quantiles* to zero. If you set this parameter to "1", because the entire range of values fits in one quantile, no quantile statistics are collected.

Related concepts:

- "Catalog statistics" on page 117
- "Catalog statistics tables" on page 124
- "Optimizer use of distribution statistics" on page 136
- "Extended examples of distribution-statistics use" on page 138

Related tasks:

- "Collecting catalog statistics" on page 120
- "Collecting distribution statistics for specific columns" on page 121

Related reference:

- "Number of Frequent Values Retained configuration parameter - num_freqvalues" on page 493
- "Number of Quantiles for Columns configuration parameter - num_quantiles" on page 494

Optimizer use of distribution statistics

The optimizer uses distribution statistics for better estimates of the cost of various possible access plans to satisfy queries.

If you do not execute RUNSTATS with the WITH DISTRIBUTION clause, the catalog statistics tables contain information only about the size of the table and the highest and lowest values in the table, the degree of clustering of the table to any of its indexes, and the number of distinct values in indexed columns.

Unless it has additional information about the distribution of values between the low and high values, the optimizer assumes that data values are evenly distributed. If data values differ widely from each other, are clustered in some parts of the range, or contain many duplicate values, the optimizer will choose a less than optimal access plan.

Consider the following example:

The optimizer needs to estimate the number of rows containing a column value that satisfies 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
```

Assume that there is an index on both C1 and 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 alternate 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'`. Because the primary cost in executing the query is usually the cost of retrieving the rows, the best plan is the plan that requires the fewest retrievals. Choosing this plan requires estimating the number of rows that satisfy each predicate.

When distribution statistics are not available but RUNSTATS has been executed against a table, the only information available to the optimizer is 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:

$$\frac{\text{KEY2 - KEY1}}{\text{HIGH2KEY - LOW2KEY}} \times \text{CARD} \qquad (1)$$

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.

Related concepts:

- "Catalog statistics" on page 117
- "Distribution statistics" on page 133
- "Extended examples of distribution-statistics use" on page 138

Related tasks:

- "Collecting distribution statistics for specific columns" on page 121

Extended examples of distribution-statistics use

To understand how the optimizer might use distribution statistics, consider first a query that contains an equality predicate of the form `C1 = KEY`.

Example for Frequent-Value Statistics

If frequent-value statistics are available, the optimizer can use these statistics to choose an appropriate access plan, as follows:

- If `KEY` is one of the N most frequent values, then the optimizer 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 CARD is the number of rows in the table, COLCARD is the cardinality of the column and 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 (`C1`) for which the frequency of the data values is as follows:

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

If frequent-value statistics based on only the most frequent value (that is, N = 1) are available, for this column, the number of rows in the table is 50 and the column cardinality is 5. For the predicate C1 = 3, exactly 40 rows satisfy it. If the optimizer assumes that data is evenly distributed, it estimates the number of rows that satisfy the predicate as 50/5 = 10, with an error of -75%. If the optimizer can use frequent-value statistics, the number of rows is estimated as 40, with no error.

Consider another example in which 2 rows satisfy the predicate C1 = 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
```

Example for Quantile Statistics

The following explanations of quantile statistics use the term “K-quantile”. 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. To computer a K-quantile, sort 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.

If quantile statistics are available, the optimizer can better estimate the number of rows that satisfy a range predicate, as illustrated by the following examples. Consider a column (C) that contains the following values:

C
0.0
5.1
6.3
7.1
8.2

C
8.4
8.5
9.1
93.6
100.0

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

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%.

If quantile statistics are available, the optimizer estimates the number of rows that satisfy this same predicate (`C <= 8.5`) by locating 8.5 as the highest value in one of the quantiles and estimating the number of rows by using the corresponding value of K, which is 7. In this case, the error is reduced to 0.

Now consider the predicate `C <= 10`. Exactly 8 rows satisfy this predicate. If the optimizer must assume a uniform data distribution and use formula (1), the number of rows that satisfy the predicate is estimated as 1, an error of -87.5%.

Unlike the previous example, the value 10 is not one of the stored K-quantiles. However, the optimizer can use quantiles to estimate 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:

```
       10 - 8.5
r_2 *= ---------- x (number of rows with value > 8.5 and <= 100.0)
       100 - 8.5
```

```
       10 - 8.5
r_2 *= ---------- x (10 - 7)
       100 - 8.5
       1.5
r_2 *= ---- x (3)
       91.5
r_2 *= 0
```

The final estimate is r_1 + r_2 *= 7, and the error is only -12.5%.

Quantiles improves the accuracy of the estimates in the above examples because 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.

Related concepts:

- "Catalog statistics" on page 117
- "Distribution statistics" on page 133
- "Optimizer use of distribution statistics" on page 136
- "Rules for updating distribution statistics manually" on page 153

Related tasks:

- "Collecting distribution statistics for specific columns" on page 121

Detailed index statistics

If you execute RUNSTATS for indexes with the DETAILED clause, you collect statistical information about indexes that allows the optimizer to estimate how many data page fetches will be required, based on various buffer-pool sizes. This additional information helps the optimizer make better estimates of the cost of accessing a table through an index.

Note: When you collect detailed index statistics, RUNSTATS takes longer and requires more memory and CPU processing. The SAMPLED DETAILED option, for which information calculated only for a statistically significant number of entries, requires 2MB of the statistics heap. Allocate an additional 488 4K pages to the *stat_heap_sz* database

configuration parameter setting for this additional memory requirement. If the heap appears to be too small, RUNSTATS returns an error before attempting to collect statistics.

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.

The DETAILED statistics provide concise information about the number of physical I/Os required to access the data pages of a table if a complete index scan is performed under different buffer sizes. As RUNSTATS scans the pages of the index, it models the different buffer sizes, and gathers estimates of how often a page fault occurs. For example, if only one buffer page is available, each new page referenced by the index results in a page fault. In a worse case, each row might reference a different page, resulting in at most the same number of I/Os as rows in the indexed table. At the other extreme, when the buffer is big enough to hold the entire table (subject to the maximum buffer size), then all table pages are read once. As a result, the number of physical I/Os is a monotone, non-increasing function of the buffer size.

The statistical information also provides finer estimates of the degree of clustering of the table rows to the index order. The less the table rows are clustered in relation to the index, the more I/Os are required to access table rows through the index. The optimizer considers both the buffer size and the degree of clustering when it estimates the cost of accessing a table through an index.

You should collect DETAILED index statistics when queries reference columns that are not included in the index. In addition, DETAILED index statistics should be used in the following circumstances:
- The table has 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 is difficult to evaluate these conditions without previous knowledge or without forcing an index scan under varying buffer sizes and then monitoring the physical I/Os that result. Probably the cheapest way to determine whether any of these situations occur is to collect the DETAILED statistics for an index, examine them, and retain them if the PAGE_FETCH_PAIRS that result are non-linear.

Related concepts:

- "Catalog statistics" on page 117
- "Catalog statistics tables" on page 124

Related tasks:

- "Collecting catalog statistics" on page 120
- "Collecting index statistics" on page 123

Sub-element statistics

If tables contain columns that contain sub-fields or sub-elements separated by blanks, and queries reference these columns in WHERE clauses, you should collect sub-element statistics to ensure the best access plans.

For example, suppose a database contains a table, DOCUMENTS, in which each row describes a document, and suppose that in DOCUMENTS there is a column called KEYWORDS that contains a list of relevant keywords relating to this document for text retrieval purposes. The values in KEYWORDS might be as follows:

```
'database simulation analytical business intelligence'
'simulation model fruit fly reproduction temperature'
'forestry spruce soil erosion rainfall'
'forest temperature soil precipitation fire'
```

In this example, each column value consists of 5 sub-elements, each of which is a word (the keyword), separated from the others by one blank.

For queries that specify LIKE predicates on such columns using the % match_all character:

```
SELECT .... FROM DOCUMENTS WHERE KEYWORDS LIKE '%simulation%'
```

it is often beneficial for the optimizer to know some basic statistics about the sub-element structure of the column.

The following statistics are collected when you execute RUNSTATS with the LIKE STATISTICS clause:

SUB_COUNT
The average number of sub-elements.

SUB_DELIM_LENGTH
The average length of each delimiter separating each sub-element, where a delimiter, in this context, is one or more consecutive blank characters.

In the KEYWORDS column example, SUB_COUNT is 5, and SUB_DELIM_LENGTH is 1, because each delimiter is a single blank character.

The DB2_LIKE_VARCHAR registry variable affects the way in which the optimizer deals with a predicate of the form:

```
COLUMN LIKE '%xxxxxx'
```

where *xxxxxx* is any string of characters; that is, any LIKE predicate whose search value starts with a % character. (It might or might not end with a % character). These are referred to as ″wildcard LIKE predicates″. For all predicates, the optimizer has to estimate how many rows match the predicate. For wildcard LIKE predicates, the optimizer assumes that the COLUMN being matched contains a series of elements concatenated together, and it estimates the length of each element based on the length of the string, excluding leading and trailing % characters.

To examine the values of the sub-element statistics, query SYSIBM.SYSCOLUMNS. For example:

```
select substr(NAME,1,16), SUB_COUNT, SUB_DELIM_LENGTH
   from sysibm.syscolumns where tbname = 'DOCUMENTS'
```

Note: RUNSTATS might take longer if you use the LIKE STATISTICS clause. For example, RUNSTATS might take between 15% and 40%, and longer on a table with five character columns, if the DETAILED and DISTRIBUTION options are not used. If either the DETAILED or the DISTRIBUTION option is specified, the overhead percentage is less, even though the absolute amount of overhead is the same. If you are considering using this option, you should assess this overhead against improvements in query performance.

Related concepts:

- “Catalog statistics” on page 117
- “Catalog statistics tables” on page 124

Related tasks:

- “Collecting catalog statistics” on page 120
- “Collecting distribution statistics for specific columns” on page 121

Catalog statistics that users can update

This section describes the catalog statistics data that users can update manually and provides guidelines for such manual changes. In some cases, this statistical data is not collected by RUNSTATS and thus must be added manually. In other cases, you might import collecte statistics data to a test database and change the collected data for a specific purpose, such as to model a production database for experiments.

Warning: In a production database, do not manually update data collected by RUNSTATS. Serious performance problems might result.

Statistics for user-defined functions

To create statistical information for user-defined functions (UDFs), you edit the SYSSTAT.FUNCTIONS catalog view. If UDF statistics are available, the optimizer can use them when it estimates costs for various access plans. The RUNSTATS utility does not collect statistics for UDFs. If statistics are not available the statistic column values are -1 and the optimizer uses default values that assume a simple UDF.

The following table provides information about the statistic columns for which you can provide estimates to improve performance:

Table 20. 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 might set the statistic columns as follows:

- INSTS_PER_INVOC: set to the estimated number of machine instructions required to:
  - Invoke EU_SHOE
  - Initialize the output string

  - Return the result.
- INSTS_PER_ARGBYTE: set to the estimated number of machine instructions required to convert the input string into a European shoe size.
- PERCENT_ARGBYTES: set to 100 indicating that the entire input string is to be converted
- INITIAL_INSTS, IOS_PER_INVOC, IOS_PER_ARGBYTE, and INITIAL_IOS: set each 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 is not found, which results in searching the entire second argument.
- The first argument is equally likely to appear anywhere within the second argument, which results in searching half of the second argument (on average) when the first argument is found.

You can use INITIAL_INSTS or INITIAL_IOS to record the estimated number of machine instructions or read/write requests that are performed only the first or last time the function is invoked, such as 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.

Related concepts:

- "Catalog statistics tables" on page 124
- "General rules for updating catalog statistics manually" on page 152

Catalog statistics for modeling and what-if planning

You can change the statistical information in the system catalogs so that it does not reflect the actual state of tables and indexes but allows you to examine various possible changes to the database for planning purposes. 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.

Do not manually update statistics on a production system. If you do, the optimizer might not choose the best access plan for production queries that contain dynamic SQL.

Requirements

You must have explicit DBADM authority for the database to modify statistics for tables and indexes and their components. 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. A DBADM can see statistics rows for all users, and can execute SQL UPDATE statements against the views defined in the SYSSTAT schema to update the values of these statistical columns.

A user without DBADM authority can see only those rows which contain statistics for objects over which they have CONTROL privilege. If you do not have DBADM authority, you can change statistics for individual database objects if you have the following privileges for each object:

- Explicit CONTROL privilege on tables. You can also update statistics for columns and indexes for these tables.
- Explicit CONTROL privilege on nicknames 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.
- Ownership of user-defined functions (UDFs)

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 manually updating catalog statistics. Arbitrary changes can seriously alter the performance of subsequent queries. Even in a non-production database that you are using for testing or modeling, you can use any of the following methods to refresh updates you applied to these tables and bring the statistics to a consistent state:

- ROLLBACK the unit of work in which the changes have been made (assuming the unit of work has not been committed).
- Use the RUNSTATS utility to 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 before you made any changes. This method is possible only if you used the *db2look* tool to capture the statistics before you made any changes.

In some cases, the optimizer may determine that some particular statistical value or combination of values is 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.

Related concepts:

- "Catalog statistics" on page 117
- "Catalog statistics tables" on page 124
- "Statistics for user-defined functions" on page 146
- "Statistics for modeling production databases" on page 149

Statistics for modeling production databases

Sometimes you may want your test system to 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*, 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 `-e` option and the `-m` option together in a script that re-creates the database and sets the statistics.

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
```

The Control Center also provides an interface to the *db2look* utility called "Generate SQL - Object Name". Using the Control Center allows 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 or z/OS 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 or z/OS objects and re-create them in a DB2® Universal Database (UDB) database. /p>

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 or z/OS 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 or z/OS 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 or z/OS statistics column.

1. Table Level Statistics.

   UDB_CARD = S390_CARDF
   UDB_NPAGES = S390_NPAGES

   There is no S390_FPAGES. However, DB2 for OS/390 or z/OS 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 or z/OS 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 or z/OS 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.

   ```
   UDB_COLVALUE = S390_COLVALUE
   UDB_VALCOUNT = S390_FrequencyF * S390_CARD
   ```

   In addition, there is no column SEQNO in DB2 for OS/390 SYSIBM.SYSCOLUMNS. Because this required for DB2 for UDB, db2look generates one automatically.

Related concepts:

- “Catalog statistics” on page 117
- “Catalog statistics tables” on page 124
- “Catalog statistics for modeling and what-if planning” on page 147
- “General rules for updating catalog statistics manually” on page 152

Related reference:

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

General rules for updating catalog statistics manually

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 updateable for a sub-table.

Related concepts:

- "Catalog statistics tables" on page 124
- "Statistics for user-defined functions" on page 146
- "Catalog statistics for modeling and what-if planning" on page 147
- "Statistics for modeling production databases" on page 149
- "Rules for updating column statistics manually" on page 153
- "Rules for updating distribution statistics manually" on page 153
- "Rules for updating table and nickname statistics manually" on page 154
- "Rules for updating index statistics manually" on page 155

Rules for updating column statistics manually

When you are updating statistics in SYSSTAT.COLUMNS, follow the guidelines below.

- HIGH2KEY and LOW2KEY (in SYSSTAT.COLUMNS) must adhere to the following rules:
  - The data type of any HIGH2KEY, LOW2KEY value must correspond to the data type 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 data type.
  - HIGH2KEY must be greater than LOW2KEY whenever there are more than 3 distinct values in the corresponding column. In the case of 3 or less distinct values in the column, HIGH2KEY can be equal to LOW2KEY.
- 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).
- The number of nulls in a column (NUMNULLS statistic in SYSSTAT.COLUMNS) cannot be greater than the cardinality of its corresponding table (CARD statistic in SYSSTAT.TABLES).
- No statistics are supported for columns with data types: LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB.

Related concepts:

- "Catalog statistics" on page 117
- "Catalog statistics tables" on page 124
- "Catalog statistics for modeling and what-if planning" on page 147
- "General rules for updating catalog statistics manually" on page 152

Rules for updating distribution statistics manually

You update distribution statistics manually only to model a production database or perform what-if tests on an artificially constructed database. Do not update distribution statistics on a production database.

Make sure that all the statistics in the catalog are consistent. Specifically, for each column, the catalog entries for the frequent data statistics and quantiles must satisfy the following constraints:

- Frequent value statistics (in the SYSSTAT.COLDIST catalog). These constraints include:

  - The values in column VALCOUNT must be unchanging or decreasing 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.
- Quantiles (in the SYSSTAT.COLDIST catalog). These constraints include:
  - The values in column COLVALUE must be unchanging or decreasing for increasing values of SEQNO
  - The values in column VALCOUNT must be 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 you do not follow these rules, the optimizer might use the wrong filter factor and cause unpredictable performance when you run the query.

Related concepts:

- "Catalog statistics" on page 117
- "Catalog statistics tables" on page 124
- "Catalog statistics for modeling and what-if planning" on page 147
- "General rules for updating catalog statistics manually" on page 152

Rules for updating table and nickname statistics manually

The only statistical values that you can update in SYSTAT.TABLES are CARD, FPAGES, NPAGES, and OVERFLOW, and for MDC tables, ACTIVE_BLOCKS. Keep in mind that:

1. CARD must be greater than or equal to 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 or 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.

Related concepts:

- “Catalog statistics” on page 117
- “Catalog statistics tables” on page 124
- “Catalog statistics for modeling and what-if planning” on page 147
- “General rules for updating catalog statistics manually” on page 152

Rules for updating index statistics manually

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.

- 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 (in SYSSTAT.COLUMNS) 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.

Related concepts:

- “Catalog statistics” on page 117
- “Catalog statistics tables” on page 124
- “Catalog statistics for modeling and what-if planning” on page 147
- “General rules for updating catalog statistics manually” on page 152

Chapter 6. Understanding the SQL compiler

When an SQL query is compiled, a number of steps are performed before the selected access plan is either executed or stored in the system catalog.

In a partitioned database environment, all of the work done on an SQL query by the SQL Compiler takes place at the database partition to which you connect. Before the compiled query runs, it is sent to all database partitions in the database.

The topics in this chapter provide more information about how the SQL compiler compiles and optimizes SQL statements.

The SQL compiler process

The SQL compiler performs several steps to produce an access plan that can be executed. These steps are shown in the following figure and described in the sections below the figure. Note that some steps occur only for queries in a federated database.

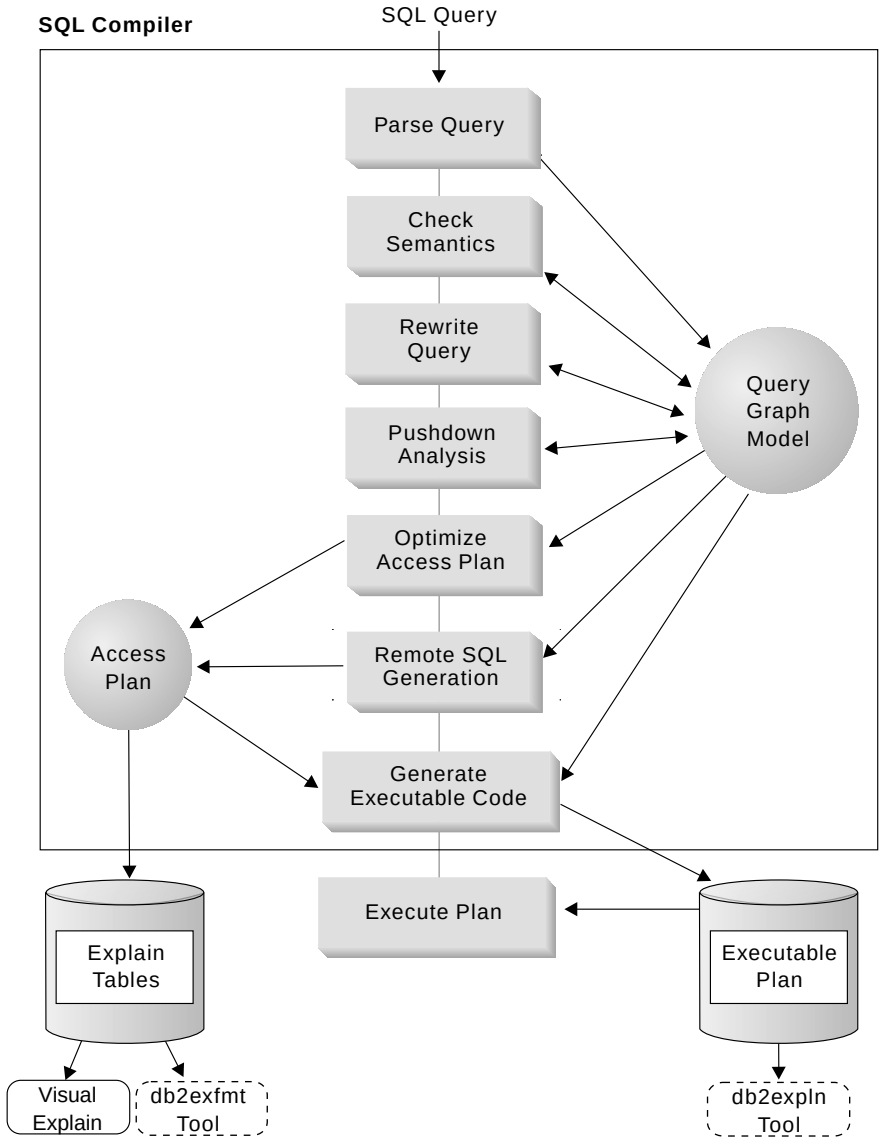


Figure 12. Steps performed by SQL compiler

Query Graph Model

The *query graph model* is an internal, in-memory database that represents the query as it is processed in the steps described below:

1. **Parse Query**

   The SQL compiler analyzes the SQL query to validate the syntax. If any syntax errors are detected, the SQL compiler stops processing and returns

the appropriate SQL error to the application that submitted the query. When parsing is complete, an internal representation of the query is created and stored in the query graph model.

2. **Check Semantics**

   The compiler ensures that there are no inconsistencies among parts of the statement. As a simple example of semantic checking, the compiler verifies that the data type of the column specified for the YEAR scalar function is a datetime data type.

   The compiler also 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 the query, including query blocks, subqueries, correlations, derived tables, expressions, data types, data type conversions, code page conversions, and partitioning keys.

3. **Rewrite Query**

   The compiler uses the global semantics stored in the query graph model to transform the query into a form that can be optimized more easily and stores the result in the query graph model.

   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*. In a partitioned database environment, the following query operations are more computationally intensive:

   - Aggregation
   - Redistribution of rows
   - Correlated subqueries, which are subqueries that contain a reference to a column of a table that is outside of the subquery.

   For some queries in a partitioned environment, decorrelation might occur as part of rewriting the query.

4. **Pushdown Analysis (Federated Databases)**

   The major task in this step is to recommend to the optimizer whether an operation can be remotely evaluated or *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.

5. **Optimize Access Plan**

   Using the query graph model as input, the optimizer portion of the compiler generates many alternative execution plans for satisfying the query. To estimate the execution cost of each alternative plan, the optimizer uses the statistics for tables, indexes, columns and functions. Then it 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 can also consider another type of pushdown operation, *aggregation and sort*, which can improve performance by pushing the evaluation of these operations to the Data Management Services component.

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.

The output of this step of the compiler is an access plan. This access plan provides the information captured in the Explain tables. The information used to generate the access plan can be captured with an explain snapshot.

6. **Remote SQL Generation (Federated Databases)**

   The final plan selected by the optimizer might consist of a set of steps that operate on a remote data source. For operations that are performed by each data source, the remote SQL generation step creates an efficient SQL statement based on the data-source SQL dialect.

7. **Generate "Executable" Code**

   In the final step, the 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 need to be computed only 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. This information is used by the *db2expln* tool.

Note: Execute RUNSTATS at appropriate intervals on tables that change often. The optimizer needs up-to-date statistical information about the tables and their data to create the most efficient access plans. Rebind your application to take advantage of updated statistics. If RUNSTATS is not executed or the optimizer suspects that RUNSTATS was executed on empty or nearly empty tables, it 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). The total number of occupied blocks is stored in the ACTIVE_BLOCKS column.

Related concepts:

- "Query rewriting methods and examples" on page 166
- "Data-access methods and concepts" on page 176
- "Predicate terminology" on page 184
- "Joins" on page 187
- "Effects of sorting and grouping" on page 204
- "Optimization strategies for intra-partition parallelism" on page 206
- "Automatic summary tables" on page 210
- "Guidelines for analyzing where a federated query is evaluated" on page 218
- "Optimization strategies for MDC tables" on page 209

Configuration parameters that affect 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. In a partitioned database, the values used for each parameter should be the same on all partitions.

Note: When you change a configuration parameter dynamically, the optimizer might not read the changed parameter values immediately because of older access plans in the package cache. To reset the package cache, execute the FLUSH PACKAGE CACHE command.

In a federated system, if the majority of your queries access nicknames, evaluate the types of queries that you send before you change your environment. For example, in a federated database the buffer pool does not cache pages from data sources, which are the DBMSs and data within the federated system. For this reason, increasing the size of the buffer does not guarantee that the optimizer will consider additional access-plan alternatives when it chooses an access plan for queries that contain nicknames. However, the optimizer might 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 might improve performance.

The following configuration parameters or factors affect the access plan chosen by the SQL compiler:

- The size of the buffer pools that you specified when you created or altered them.

  When the optimizer chooses the access plan, it considers the I/O cost of fetching pages from disk to the buffer pool and estimates the number of

I/Os required to satisfy a query. The estimate includes a prediction of buffer-pool usage, because 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 and, on partitioned databases, the BUFFERPOOLDBPARTITION system catalog tables.

The I/O costs of reading the tables can have an impact on:
– How two tables are joined
– Whether an unclustered index will be used to read the data

- Default Degree (dft_degree)

  The `dft_degree` configuration parameter specifies parallelism by providing a 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)

  Although you can specify a query optimization class when you compile SQL queries, you might set a default optimization degree.

  Note: Intra-parallel processing does not occur unless you enable it by setting the *intra_parallel* database configuration parameter.

- Average Number of Active Applications (avg_appls)

  The SQL optimizer uses the *avg_appls* parameter to help estimate how much of the buffer pool might be available at run-time for the access plan chosen. Higher values for this parameter can influence the optimizer to choose access plans that are more conservative in buffer pool usage. If you specify a value of 1, the optimizer considers that the entire buffer pool will be available to the application.

- Sort Heap Size (sortheap)

  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, which might be written to disk. When all the sort passes are complete, these sorted subsets are merged into a single sorted set of rows. 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.

  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) and Maximum Percent of Lock List Before Escalation (maxlocks)

  When the isolation level is **repeatable read (RR)**, the SQL optimizer considers the values of the *locklist* and *maxlocks* parameters to determine whether row level locks might be escalated to a table level lock. If the optimizer estimates that lock escalation will occur for a table access, then it chooses a table level lock for the access plan, instead of incurring the overhead of lock escalation during the query execution.

- CPU Speed (cpuspeed)

  The SQL optimizer uses the CPU speed to estimate the cost of performing certain operations. CPU cost estimates and various I/O cost estimates help 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. Do not adjust this parameter unless you are modelling a production environment on a test system or assessing the impact of a hardware change. Using this parameter to model a different hardware environment allows you to find out the access plans that might be chosen for that environment. To have DB2 recompute the value of this automatic configuration parameter, set it to -1.

- Statement Heap Size (stmtheap)

  Although the size of the statement heap does not influence the optimizer in choosing different access paths, it can affect the amount of optimization performed for complex SQL statements.

  If the *stmtheap* parameter is not set large enough, you might receive an SQL warning indicating that there is not enough memory available to process the statement. For example, SQLCODE +437 (SQLSTATE 01602) might indicate that the amount of optimization that has been used to compile a statement is less than the amount that you requested.

- Maximum Query Degree of Parallelism (max_querydegree)

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

- Communications Bandwidth (comm_bandwidth)

  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.

Related concepts:

- "The SQL compiler process" on page 159
- "Data-access methods and concepts" on page 176
- "Optimization strategies for intra-partition parallelism" on page 206
- "Optimization strategies for MDC tables" on page 209

Related reference:

- "Maximum Query Degree of Parallelism configuration parameter - max_querydegree" on page 505
- "Communications Bandwidth configuration parameter - comm_bandwidth" on page 513
- "Sort Heap Size configuration parameter - sortheap" on page 405
- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
- "Statement Heap Size configuration parameter - stmtheap" on page 409
- "CPU Speed configuration parameter - cpuspeed" on page 513
- "Average Number of Active Applications configuration parameter - avg_appls" on page 440
- "Default Degree configuration parameter - dft_degree" on page 491

Query rewriting

This section the ways in which the optimizer can rewrite queries to improve performance.

Query rewriting methods and examples

During the rewrite query stage, the SQL compiler transforms SQL statements into forms that can be optimized more easily, and as a result, can improve the possible access paths. Rewriting queries is particularly important for very complex queries, including those queries with many subqueries or many joins. Query generator tools often create these types of very complex queries.

To influence the number of query rewrite rules that are applied to an SQL statement, change the optimization class. To see some of the results of the query rewrite, use the Explain facility or Visual Explain.

Queries might be rewritten in any of the following three primary ways:

- **Operation merging**

To construct the query so that it has the fewest number of operations, especially SELECT operations, the SQL compiler rewrites queries to merge query operations. The following examples illustrate some of the operations that can be merged:

– Example - View Merges

  A SELECT statement that uses views can restrict the join order of the table and can also introduce redundant joining of tables. If the views are merged during query rewrite, these restrictions can be lifted.

– Example - Subquery to Join Transforms

  If a SELECT statement contains a subquery, selection of order processing of the tables might be restricted.

– Example - Redundant Join Elimination

  During query rewrite, redundant joins can be removed to simplify the SELECT statement.

– Example - Shared Aggregation

  When the query uses different functions, rewriting can reduce the number of calculations that need to be done.

- **Operation movement**

  To construct the query with the minimum number of operations and predicates, the compiler rewrites queries to move query operations. The following examples illustrate some of the operations that can be moved:

  – Example - DISTINCT Elimination

    During query rewrite, the optimizer can move the point at which the DISTINCT operation is performed, to reduce the cost of this operation. In the extended example provided, the DISTINCT operation is removed completely.

  – Example - General Predicate Pushdown

    During query rewrite, the optimizer can change the order of applying predicates so that more selective predicates are applied to the query as early as possible.

  – Example - Decorrelation

    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, or reducing the number of broadcasts, or both, is an objective of query rewriting.

- **Predicate Translation**

  The SQL compiler rewrites queries to translate existing predicates to more optimal predicates for the specific query. The following examples illustrate some of the predicates that might be translated:

  – 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 - ON to IN Transformations

During query rewrite, an OR predicate can be translated into an IN predicate for a more efficient access plan. The SQL compiler can also translate an IN predicate into an OR predicate if this transformation would create a more efficient access plan.

Related concepts:

- "Compiler rewrite example: view merges" on page 168
- "Compiler rewrite example: DISTINCT elimination" on page 171
- "Compiler rewrite example: implied predicates" on page 173
- "Column correlation for multiple predicates" on page 174

Compiler rewrite 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.

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.

```
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 users know 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.

Related concepts:

- "The SQL compiler process" on page 159
- "Query rewriting methods and examples" on page 166

Compiler rewrite 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 is not rewritten, the optimizer would need to build a plan with the necessary processing, such as a sort, 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.

Related concepts:

- "The SQL compiler process" on page 159
- "Query rewriting methods and examples" on page 166

Compiler rewrite example: 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 adds 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 also derives 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 the employees who work in those departments.

```
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 adds 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 allows the query to be processed more efficiently:

```
SELECT *
  FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

Note: In some cases, the database manager might convert an IN predicate to a set of OR clauses so that index ORing might be performed.

Related concepts:

- "The SQL compiler process" on page 159
- "Query rewriting methods and examples" on page 166

Column correlation for multiple predicates

Your applications might contain queries that are constructed with joins such that more than one join predicate joins two tables. This is not unusual when queries need to determine relationships between similar, related columns in different tables.

For example, consider a manufacturer who 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 optimizer chooses a plan for executing this query, it calculates how selective each of the two predicates is. It assumes that they are independent, which means 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 estimate the overall selectivity of the pair of predicates by using catalog statistic information for each table on the number of levels of elasticity and the number of different colors. Based on this estimate, 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 that are different from the elastic ones. Then the combined selectivity of the predicates eliminates fewer rows so the query will return more rows. Consider the extreme case, in which 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 might no longer choose the best plan. For example, it might choose a nested loop join plan when 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.

The DB2® UDB optimizer attempts to detect and compensate for correlation of join predicates if you use the two following rules:

For example, in elasticity example above, you might define a unique index covering either:

```
PRODUCT.COLOR, PRODUCT.ELASTICITY
```

or

```
RAWMATERIAL.COLOR, RAWMATERIAL.ELASTICITY
```

or both.

For the optimizer to detect correlation, the non-include columns of this index must be only the correlated columns. The index may also contain include columns to allow index-only scans. If there are more than two correlated columns in join predicates, make sure that you define the unique index to cover all of them. In many cases, the correlated columns in one table are its primary key. Because a primary key is always unique, you do not need to define another unique index.

After creating appropriate indexes, ensure that statistics on tables are up to date and that they have not been manually altered from the true values for any reason, such as to attempt to influence the optimizer.

The optimizer uses the information in the FIRSTnKEYCARD and FULLKEYCARD columns of the unique index statistics table 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.

Correlation of simple equal predicates

In addition to JOIN predicate correlation, the optimizer also manages 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, YEAR, COLOR, and STYLE, such as sedan, station wagon, sports-utility vehicle. Because almost every manufacturer makes the same standard colors available on each of their models and styles, year after year, predicates on COLOR are likely to be independent of those on MAKE, MODEL, STYLE, or 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 uses the statistical information about the index to determine the combined number of distinct values and adjust the selectivity or cardinality estimation for correlation between the two columns. If such predicates are not join predicates, the optimizer does not need a unique index to make the adjustment.

Related concepts:

- "The SQL compiler process" on page 159
- "Query rewriting methods and examples" on page 166

Data access methods

This section describes the methods the optimizer can choose to access data required by a query.

Data-access methods and concepts

When it compiles an SQL statement, the SQL optimizer estimates the execution cost of different ways of satisfying the query. Based on its estimates, the optimizer selects an 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 to access data in a table:

- Scanning the entire table sequentially
- Locating specific table rows by first accessing an index on the table

To produce the results that the query requests, rows are selected depending on the terms of the predicate, which are usually stated in a WHERE clause.

The selected rows in accessed tables are joined to produce the result set, and the result set might be further processed by grouping or sorting the output.

Related concepts:

- "The SQL compiler process" on page 159
- "Data access through index scans" on page 177
- "Types of index access" on page 181
- "Index access and cluster ratios" on page 183

Data access through index scans

An *index scan* occurs when the database manager accesses an index for any of the following reasons:

- To narrow 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.
- To order the output.
- To retrieve the requested column data directly. If all of the requested data is in the index, the indexed table does not need to be accessed. This is known as an *index-only access*.

If indexes are created with the ALLOW REVERSE SCANS option, scans may also be performed in the direction opposite to that with which they were defined.

Index scans occur in the following circumstances:

- To delimit a range
- To test predicate inequality
- To order data

Note: The optimizer chooses a table scan if no appropriate index has been created or if an index scan would be more costly. An index scan might be more costly when the table is small the index-clustering ratio is low, or the query requires most of the table rows. To find out whether the access plan uses a table scan or an index scan, use the SQL Explain facility.

Index Scans to Delimit a Range

To determine 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 equality and 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 in the following cases:

- Tests for equality against a constant, a host variable, an expression that evaluates to a constant, or a keyword
- Tests for "IS NULL" or "IS NOT NULL"
- Tests for equality against a basic subquery, which is a subquery 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).
- Tests for strict and inclusive inequality.

The following examples illustrate when an index might be used to limit a range:

- Consider an index with the following definition:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

  In this case, the following predicates might be used to limit 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 WHERE clause, the predicates do not have to be specified in the same order as the key columns appear in the index. Although the examples use host variables, other variables such as parameter markers, expressions, or constants would have the same effect.

- Consider a single index created using the ALLOW REVERSE SCANS parameter. Such indexes support scans in the direction defined when the index was created as well as in the opposite or reverse direction. The statement might 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.

Index Scans to Test Inequality

Certain inequality predicates can delimit the range of an index scan. There are two types of inequality predicates:

- Strict inequality predicates

  The strict inequality operators used for range delimiting predicates are greater than (>) and less than (<).

  Only one column with strict inequality predicates is considered for delimiting a range for an index scan. 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 that 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.

Index Scans to Order Data

If the query requires output in sorted order, an index might 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", which is any expression that evaluates to a constant. In this case the ordering column can be other than the first index key columns.

Consider the following query:

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY MGR
```

For this query, the index might be used to order the rows because NAME and DEPT will always be the same values and will thus be ordered. That is, 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 a sort-order requirement. Consider 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 because the IX0 index ensures that PROJNO is unique. This uniqueness ensures that there is only one PROJNAME value for each PROJNO value.

Related concepts:

- "Data-access methods and concepts" on page 176
- "Index structure" on page 31
- "Types of index access" on page 181
- "Index access and cluster ratios" on page 183

Types of index access

In some cases, the optimizer might find that all data that a query requires from a table can be retrieved from an index on the table. In other cases, the optimizer might use more than one index to access tables.

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
```

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'
```

Often, however, required columns that do not appear in the index. To obtain the data for these columns, the table rows must be read. To allow the optimizer to choose an index-only access, create a unique index with include columns. For example, consider the following index definition:

```
CREATE UNIQUE INDEX IX1 ON EMPLOYEE
   (NAME ASC)
    INCLUDE (DEPT, MGR, SALARY, YEARS)
```

This index enforces uniqueness of the NAME column and also stores and maintains data for DEPT, MGR, SALARY, and YEARS columns, which allows the following query to be satisfied by accessing only the index:

```
SELECT NAME, DEPT, MGR, SALARY
   FROM EMPLOYEE
   WHERE NAME='SMITH'
```

When you consider adding INCLUDE columns to an index, however, consider whether the additional storage space and maintenance costs are justified. If queries that can be satisfied by reading only such an index are rarely executed, the costs might not be justified.

Multiple Index Access

The optimizer can choose to scan multiple indexes on the same table to satisfy the predicates of a WHERE clause. For example, consider the following two index definitions:

```
INDEX IX2:  DEPT    ASC
INDEX IX3:  JOB     ASC,
            YEARS   ASC
```

The following predicates can be satisfied by using the two indexes:

```
WHERE DEPT = :hv1
  OR (JOB   = :hv2
  AND YEARS >= :hv3)
```

Scanning index IX2 produces a list of row IDs (RIDs) that satisfy the `DEPT = :hv1` predicate. Scanning index IX3 produces a list of RIDs satisfying the `JOB = :hv2 AND YEARS >= :hv3` predicate. These two lists of RIDs are combined and duplicates removed before the table is accessed. This is known as *index ORing*.

Index ORing may also be used for predicates specified in the IN clause, as in the following example:

```
WHERE DEPT IN (:hv1, :hv2, :hv3)
```

Although the purpose of index ORing is to eliminate duplicate RIDs, the objective of *index ANDing* is to find common RIDs. Index ANDing might occur with applications that create multiple indexes on corresponding columns in 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.

To realize the performance benefits of dynamic bitmaps when scanning multiple indexes, it may be necessary to change the value of the sort heap size (*sortheap*) database configuration parameter, and the sort heap threshold (*sheapthres*) database manager configuration parameter.

Additional sort heap space is required when dynamic bitmaps are used in access plans. When *sheapthres* is set to be relatively close to *sortheap* (that is, less than a factor of two or three times per concurrent query), dynamic bitmaps with multiple index access must work with much less memory than the optimizer anticipated. The solution is to increase the value of *sheapthres* relative to *sortheap*.

Note: The optimizer does not combine index ANDing and index ORing in accessing a single table.

Related concepts:

- "Advantages and disadvantages of indexes" on page 294
- "Data-access methods and concepts" on page 176
- "Data access through index scans" on page 177
- "Index access and cluster ratios" on page 183

Index access and cluster ratios

When it chooses an access plan, the optimizer estimates the number of I/Os required to fetch required pages from disk to the buffer pool. 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, information from the system catalog tables (SYSCAT.INDEXES) helps the optimizer estimate I/O cost of reading data pages into the buffer pool. It uses information from the following columns in the SYSCAT.INDEXES table:

- CLUSTERRATIO information indicates the degree to which the table data is clustered in relation to this index. The higher the number, the better rows are ordered in index key sequence. If table rows are in close to index-key sequence, rows can be read from a data page while the page is in the buffer. If the value of this column is -1, the optimizer uses PAGE_FETCH_PAIRS and CLUSTERFACTOR information if it is available.
- PAGE_FETCH_PAIRS contains pairs of numbers that model the number of I/Os required to read the data pages into buffer pools of various sizes together with CLUSTERFACTOR information. Data is collected for these columns only if you execute RUNSTATS on the index with the DETAILED clause.

If index clustering statistics are not available, the optimizer uses default values, which assume poor clustering of the data to the index.

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.

When you reorganize an table, you can specify an index that will be used to cluster the rows and attempt to preserve this characteristic during insert processing. Because updates and inserts may make the table less well clustered in relation to the index, you might need to periodically reorganize the table. To reduce the frequency of reorganization on a table that has frequent changes due to INSERTs, UPDATEs, and DELETES, use the PCTFREE parameter when you alter a table. This allows for additional inserts to be clustered with the existing data.

Related concepts:

- "Index performance tips" on page 299
- "Types of index access" on page 181

Predicate terminology

A user application requests a set of rows from the database with an SQL statement that specifies qualifiers for the specific rows to be returned as the result set. These qualifiers usually appear in the WHERE clause of the query. Such qualifiers are called *predicates*. Predicates can be grouped into four categories that are determined by how and when the predicate is used in the evaluation process. The 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.

Note: *SARGable* refers to a term that can be used as a *s*earch *arg*ument.

The following table summarizes the predicate categories. Subsequent sections describe each category in more detail.

Table 21. 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

Range-Delimiting and Index-SARGable Predicates

Range delimiting predicates limit the scope of an index scan. They provide start and stop key values for the index search. Index SARGable predicates cannot limit the scope of a search, but can be evaluated from the index because the columns involved in the predicate are part of the index key. For example, consider the following index:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

Consider also a query that contains the following WHERE clause:

```
WHERE  NAME  = :hv1
 AND   DEPT  = :hv2
 AND   YEARS > :hv5
```

The first two predicates (NAME = :hv1, DEPT = :hv2) are range-delimiting predicates, while YEARS > :hv5 is an index SARGable predicate.

The optimizer uses the index data when it evaluates these predicates instead of reading the base table. These *index SARGable* predicates reduce the set of rows that need to be read from the table, but they do not affect the number of index pages that are accessed.

Data SARGable Predicates

Predicates that cannot be evaluated by the index manager, but can be evaluated by data management services are called *data SARGable* predicates. These predicates usually require accessing individual rows from a table. If required, Data Management Services 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, consider a single index defined on the PROJECT table:

```
INDEX IX0:  PROJNO ASC
```

For the following query, then, 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 require more I/O costs than accessing a table. They might have the following characteristics:

- Use correlated subqueries
- Use quantified subqueries, which contain ANY, ALL, SOME, or IN clauses
- Read LONG VARCHAR or LOB data, which is stored in a file that is separate from the table

Such predicates are evaluated by Relational Data Services.

Sometimes predicates that are applied only to the index must be reapplied when the data page is accessed. For example, access plans that use index ORing or index ANDing always reapply the predicates as residual predicates when the data page is accessed.

Related concepts:

- "The SQL compiler process" on page 159

Join methods and strategies

This section describes how the optimizer joins tables as required to return results to a query and explains the methods and strategies that the optimizer employs.

Joins

A *join* is the process of combining information from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the joining criterion.

For example, consider 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

To join Table1 and Table2 where the ID columns have the same values, use the following SQL statement:

```
SELECT PROJ, x.PROJ_ID, NAME
  FROM TABLE1 x, TABLE2 y
  WHERE x.PROJ_ID = y.PROJ_ID
```

This query yields 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

Depending on the existence of a join predicate, as well as various costs involved as determined by table and index statistics, the optimizer chooses one of the following join methods:

- Nested-loop join
- Merge join
- Hash join

When two tables are joined, one table is selected as the outer table and the other as the inner. The outer table is accessed first and is scanned only once. Whether the inner table is scanned multiple times depends on the type of join and the indexes that are present. Even if a query joins more than two tables,

the optimizer joins only two tables at a time. If necessary, temporary tables are created to hold intermediate results.

You can provide explicit join operators, such as INNER or LEFT OUTER JOIN to determine how tables are used in the join. Before you alter a query in this way, however, you should allow the optimizer to determine how to join the tables. Then analyze query performance to decide whether to add join operators.

Related concepts:

- "Join methods" on page 188
- "Join strategies in partitioned databases" on page 196
- "Join methods in partitioned databases" on page 198
- "Join information" on page 620

Join methods

The optimizer can choose one of three basic join strategies when queries require tables to be joined.

- Nested-loop join
- Merge join
- Hash join

These methods are described in the following sections.

Nested-Loop Join

A nested-loop join is performed in one of the following two ways:

- Scanning the inner table for each accessed row of the outer table

  For example, consider that column A in tables T1 and T2 have the following values:

Outer table T1: column A	Inner table T2: column A
2	3
3	2
3	2
	3
	1

  To perform a nested-loop join, the database manager performs the following steps:

  1. Read the first row from T1. The value for A is "2"
  2. Scan T2 until a match ("2") is found, and then join the two rows

3. Scan T2 until the next match (“2”) is found, and then join the two rows
4. Scan T2 to the end of the table
5. Go back to T1 and read the next row (“3”)
6. Scan T2, starting at the first row, until a match (“3”) is found, and then join the two rows
7. Scan T2 until the next match (“3”) is found, and then join the two rows
8. Scan T2 to the end of the table
9. Go back to T1 and read the next row (“3”)
10. Scan T2 as before, joining all rows which match (“3”).

- Performing 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. Consider the following examples:

  ```
  OUTER.C1 + OUTER.C2 <= INNER.C1
  OUTER.C4 < INNER.C3
  ```

  This method might 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 it evaluates a nested loop join, the optimizer also decides whether to sort the outer table before performing the join. If it orders the outer table, based on the join columns, the number of read operations to access pages from disk for the inner table might be reduced, because they are more likely to be be in the buffer pool already. If the join uses a highly clustered index to access the inner table and if the outer table has been sorted, the number of index pages accessed might be minimized.

In addition, if the optimizer expects that the join will make a later sort more expensive, it might also choose to perform the sort before the join. A later sort might 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. A merge join cannot be used if the join column is a LONG field column or a large object (LOB) column.

In a merge join, the joined tables are scanned at the same time. The outer table of the merge join is scanned only once. The inner table is also scanned once unless repeated values occur in the outer table. If there are repeated values occur, a group of rows in the inner table might 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

To perform a merge join, the database manager performs the following steps:

1. Read the first row from T1. The value for A is "2".
2. Scan T2 until a match is found, and then join the two rows.
3. Keep scanning T2 while the columns match, joining rows.
4. When the "3" in T2 is read, go back to T1 and read the next row.
5. The next value in T1 is "3", which matches T2, so join the rows.
6. Keep scanning T2 while the columns match, joining rows.
7. The end of T2 is reached.
8. 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

A hash join requires one or more predicates of the form table1.columnX = table2.columnY, 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, the designated INNER table is scanned and the rows copied into memory buffers drawn from the sort heap specified by the *sortheap* database configuration parameter. The memory buffers are divided into partitions based on a hash value that is computed on the columns of the join predicates. If the size of the INNER table exceeds the available sort heap space, buffers from selected partitions are written to temporary tables.

When the inner table has been processed, the second, or OUTER, table is scanned and its rows are matched to rows from the INNER table by first comparing the hash value computed for the columns of the join predicates. If the hash value for the OUTER row column matches the hash value of the INNER row column, the actual join predicate column values are compared.

OUTER table rows that correspond to partitions not written to a temporary table are matched immediately with INNER table rows in memory. 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 values of their rows are matched, and the join predicates are checked.

For the full performance benefits of hash join, you might need to change the value of the *sortheap* database configuration parameter and the *sheapthres* database manager configuration parameter.

For the full performance benefits of hash joins, you might need to change the value of the *sortheap* database configuration parameter and the *sheapthres* database manager configuration parameter.

Hash-join performance is best if you can avoid hash loops and overflow to disk. To tune hash-join performance, estimate the maximum amount of memory available for *sheapthres*, then tune the **sortheap** parameter. Increase its setting until you avoid as many hash loops and disk overflows as possible, but do not reach the limit specified by the *sheapthres* parameter.

Increasing the *sortheap* value should also improve performance of queries that have multiple sorts.

Related concepts:

- "Joins" on page 187
- "Join strategies in partitioned databases" on page 196
- "Join methods in partitioned databases" on page 198
- "Join information" on page 620

Related reference:

- "Sort Heap Size configuration parameter - sortheap" on page 405
- "Sort Heap Threshold configuration parameter - sheapthres" on page 406

Strategies for selecting optimal joins

The optimizer uses various methods to select an optimal join strategy for a query. Among these methods are the following search strategies, which are determined by the optimization class of the query:

- 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 is not changed during further optimization
  - Might miss the best access plan when joining many tables. If your query joins only two or three tables, the access plan chosen by the greedy join enumeration is the same as the access plan chosen by dynamic programming join enumeration. This is particularly true if the query has many join predicates on the same column, either explicitly specified, or implicitly generated through predicate transitive closure.
- Dynamic programming join enumeration
  - Space and time requirements increase exponentially as the number of joined tables increases
  - Efficient and exhaustive search for best access plan
  - Similar to the strategy used by DB2® for OS/390 or z/OS.

The join-enumeration algorithm is an important determinant of the number of plan combinations that the optimizer explores.

Star-Schema Joins

The tables referenced in a query are almost always related by join predicates. If two tables are joined without a join predicate, the Cartesian product of the two tables is formed. In a Cartesian product, 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 usually very large. Since such a plan is unlikely to perform well, the optimizer avoids even determining the cost of such an access plan.

The only exceptions occur when the optimization class is set to 9 or in the special case of star schemas. A *star schema* contains a central table called the fact table and the other tables are called dimension tables. The dimension tables all have only a single join that attaches them to the fact table, regardless of the query. Each dimension table contains additional values that expand information about a particular column in the fact table. A typical query consists of multiple local predicates that reference values in the dimension tables and contains join predicates connecting the dimension tables to the fact table. For these queries it might be beneficial to compute 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 can recognize queries against databases designed with star schemas that have at least two dimension tables and can increase the search space to

include possible plans that compute the Cartesian product of dimension tables. If the plan that computes the Cartesian products has the lowest estimated cost, it is selected by the optimizer.

The star schema join strategy discussed above assumes that primary key indexes are used in the join. Another scenario involves foreign key indexes. If the foreign key columns in the fact table are single-column indexes and there is a relatively high selectivity across all dimension tables, the following star join technique can be used:

1. Process each dimension table 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. Use AND predicates against the previous bitmap for each bitmap.
3. Determine the surviving RIDs after processing the last bitmap.
4. Optionally sort these RIDs.
5. Fetch a base table row.
6. Rejoin the fact table with each of its dimension tables, accessing the columns in dimension tables that are needed for the SELECT clause.
7. Reapply the residual predicates.

This technique does not require multi-column indexes. Explicit referential-integrity constraints between the fact table and the dimension tables are not required, although the relationship between the fact table and the dimension tables should actually be related in this way.

The dynamic bitmaps created and used by star join techniques require sort heap memory, the size of which is specified by the Sort Heap Size (*sortheap*) database configuration parameter.

Composite Tables

When the result of joining a pair of tables is a new table known as a *composite* table, this table usually becomes the outer table of another join with another inner table. This is known as a “composite outer” join. In some cases, particularly when using the greedy-join enumeration technique, it is useful to make the result of joining two tables the inner table of a later join. When the inner table of a join consists of the result of joining two or more tables, this plan is a “composite inner” join. For example, consider 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 might 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 table and the second join result as the inner table. 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 inner joins are allowed with optimization classes 5, 7, and 9.

Related concepts:

- "Joins" on page 187
- "Join methods" on page 188
- "Join strategies in partitioned databases" on page 196
- "Join methods in partitioned databases" on page 198

Replicated materialized-query tables in partitioned databases

Replicated materialized query tables improve performance of frequently executed joins in a partitioned database environment by allowing the database to manage precomputed values of the table data.

Consider an example of a query and a replicated materialized table. 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 database partition group.

Create a replicated materialized query 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;
```

To update the content of the replicated materialized query table, run the following statement:

```
REFRESH TABLE R_EMPLOYEE;
```

Note: After using the REFRESH statement, you should run RUNSTATS on the replicated table as you would any other table.

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 uses the R_EMPLOYEE table, which is replicated on each of the database partitions where the SALES tables is stored. 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.

Replicated materialized query tables in collocated joins

Replicated materialized query tables can also assist in the collocation of joins. For example, if a star schema contains a large fact table spread across twenty nodes, the joins between the fact table and the dimension tables are most efficient if these tables are collocated. If all of the tables are in the same database partition group, at most one dimension table is partitioned correctly for a collocated join. The other dimension tables cannot be used in a collocated join because the join columns on the fact table do not correspond to the partitioning key of the fact table.

Consider a table called FACT (C1, C2, C3, ...) partitioned on C1; and a table called DIM1 (C1, dim1a, dim1b, ...) partitioned on C1; and a table called DIM2 (C2, dim2a, dim2b, ...) partitioned on C2; and so on.

In this case, you see that the join between FACT and DIM1 is perfect because the predicate DIM1.C1 = FACT.C1 is collocated. Both of these tables are partitioned on column C1.

However, the join between DIM2 with the predicate WHERE DIM2.C2 = FACT.C2 cannot be collocated because FACT is partitioned on column C1 and not on column C2. In this case, you might replicate DIM2 in the database partition group of the fact table so that the join occurs locally on each partition.

Note: The replicated materialized query tables discussion here is related to intra-database replication. Inter-database replication is concerned with subscriptions, control tables, and data located in different databases and on different operating systems.

When you create a replicated materialized query table, the source table can be a single-node table or a multi-node table in a database partition group. In most cases, the replicated table is small and can be placed in a single-node database partition group. You can limit the data to be replicated by specifying only a subset of the columns from the table or by specifying the number of rows through the predicates used, or by using both methods. The data capture option is not required for replicated materialized query tables to function.

A replicated materialized query table can also be created in a multi-node database partition group so that copies of the source table are created on all of the partitions. Joins between a large fact table and the dimension tables are more likely to occur locally in this environment that if you broadcast the source table to all partitions.

Indexes on replicated tables are not created automatically. You can create indexes that are different from those on the source table. However, to prevent constraint violations that are not present on the source tables, you cannot create unique indexes or put constraints on the replicated tables. Constraints are disallowed even if the same constraint occurs on the source table.

Replicated tables can be referenced directly in a query, but you cannot use the NODENUMBER() predicate with a replicated table to see the table data on a particular partition.

Use the EXPLAIN facility to see if a replicated materialized query table was used by the access plan for a query. Whether the access plan chosen by the optimizer uses the replicated materialized query table depends on the information that needs to be joined. The optimizer might not use the replicated materialized query table if the optimizer determines that it would be cheaper to broadcast the original source table to the other partitions in the database partition group.

Related concepts:

- "Joins" on page 187

Join strategies in partitioned databases

In some ways, join strategies are different in a partitioned database than in a non-partitioned database. Additional techniques can be applied to standard join methods to improve performance.

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.

Table Queues

The descriptions of join techniques in a partitioned database use the following terminology:

- table queue

  A mechanism for transferring rows between database partitions, or between processors in a single partition database.
- directed table queue

  A table queue in which rows are hashed to one of the receiving database partitions.
- broadcast table queue

  A table queue in which rows are sent to all of the receiving database partitions, but are not hashed.

A table queue is used in the following circumstances:

- 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 passes 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.

Related concepts:

- "Joins" on page 187
- "Join methods" on page 188
- "Join methods in partitioned databases" on page 198

Join methods in partitioned databases

The following figures illustrate join methods in a partitioned database.

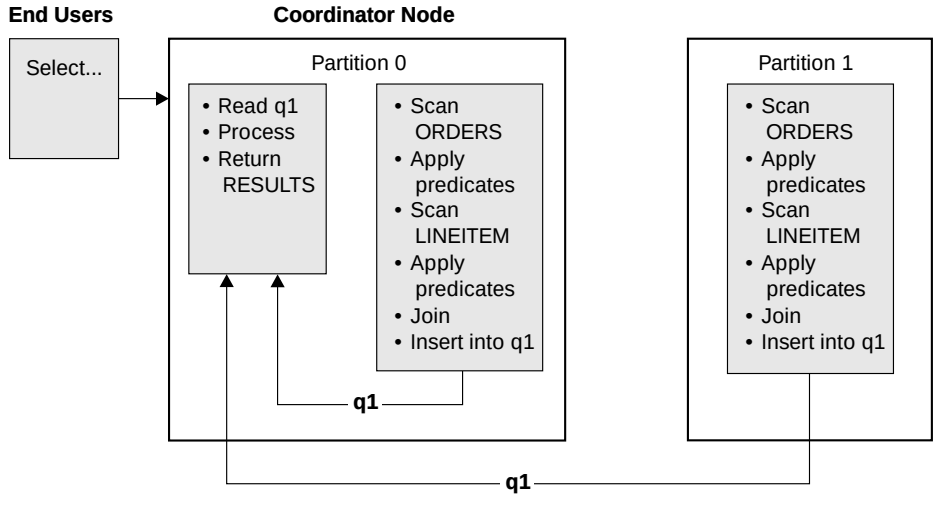
Note: In the diagrams q1, q2, and q3 refer to table queues in the examples. The tables that are referenced are divided across two database partitions for the purpose of these scenarios. The arrows indicate the direction in which the table queues are sent. The coordinator node is partition 0.

Collocated Joins

A collocated join occurs locally on the partition where the data resides. The partition sends the data to the other partitions after the join is complete. 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 equality join predicates.

The following figure provides an example.

Note: Replicated materialized query tables enhance the likelihood of collocated joins.



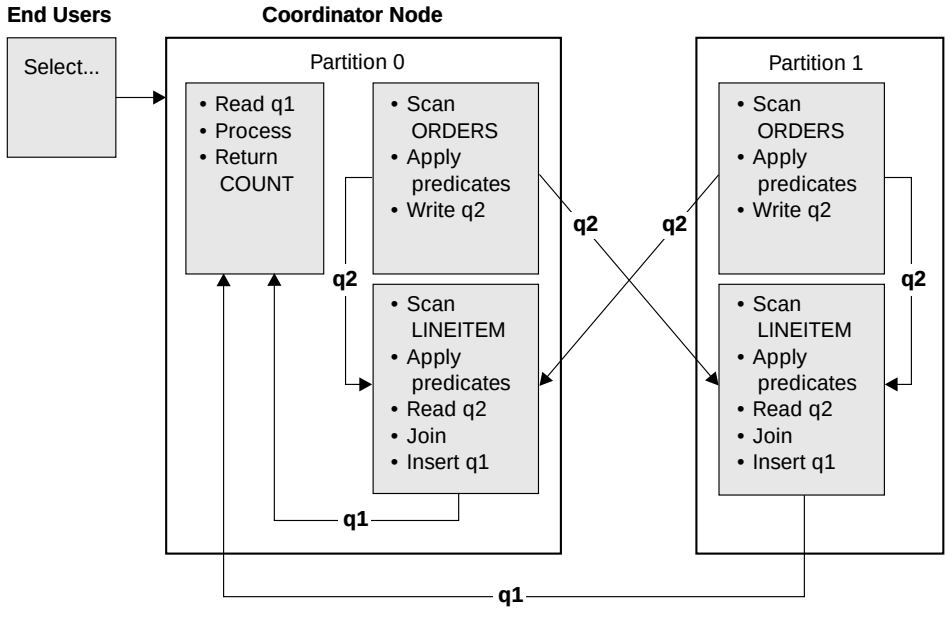
Both the LINEITEM and ORDERS tables are partitioned on the ORDERKEY column. The join is done locally at each database partition. In this example, the join predicate is assumed to be:

ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 13. Collocated Join Example

Broadcast Outer-Table Joins

Broadcast outer-table joins are a parallel join strategy that can be used if there are no equality join predicates between the joined tables. It can also be used in other situations in which it is the most cost-effective join method. For example, a broadcast outer-table join might occur when there is one very large table and one very small table, neither of which is partitioned on the join predicate columns. Instead of partitioning both tables, it might be cheaper to broadcast the smaller table to the larger table. The following figures provide an example.

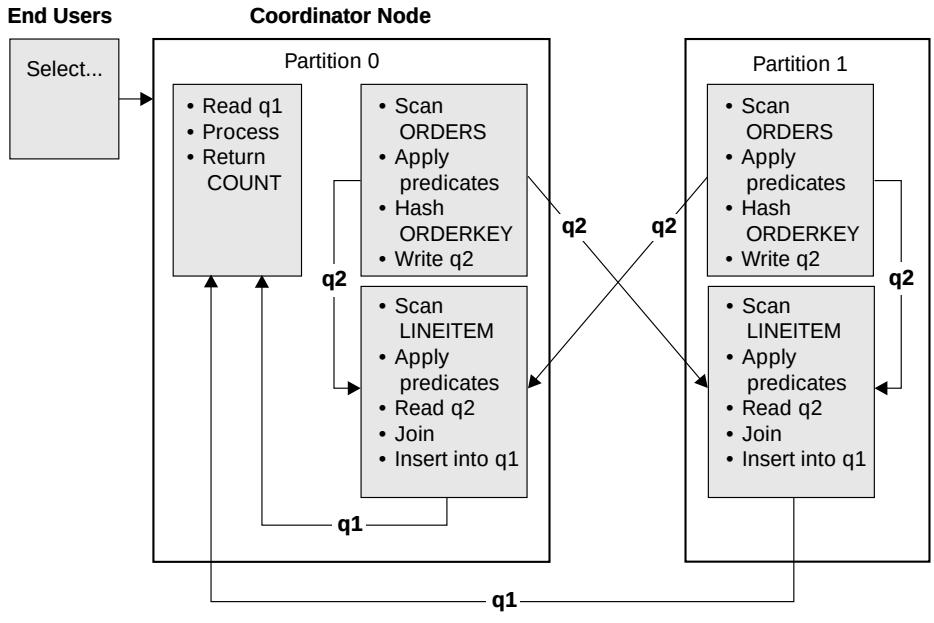


The ORDERS table is sent to all database partitions that have the LINEITEM table. Table queue q2 is broadcast to all database partitions of the inner table.

Figure 14. Broadcast Outer-Table Join Example

Directed Outer-Table Joins

In the directed outer-table join strategy, each row of the outer table is sent to one partition of the inner table, based on the partitioning attributes of the inner table. The join occurs on this database partition. The following figure provides an example.

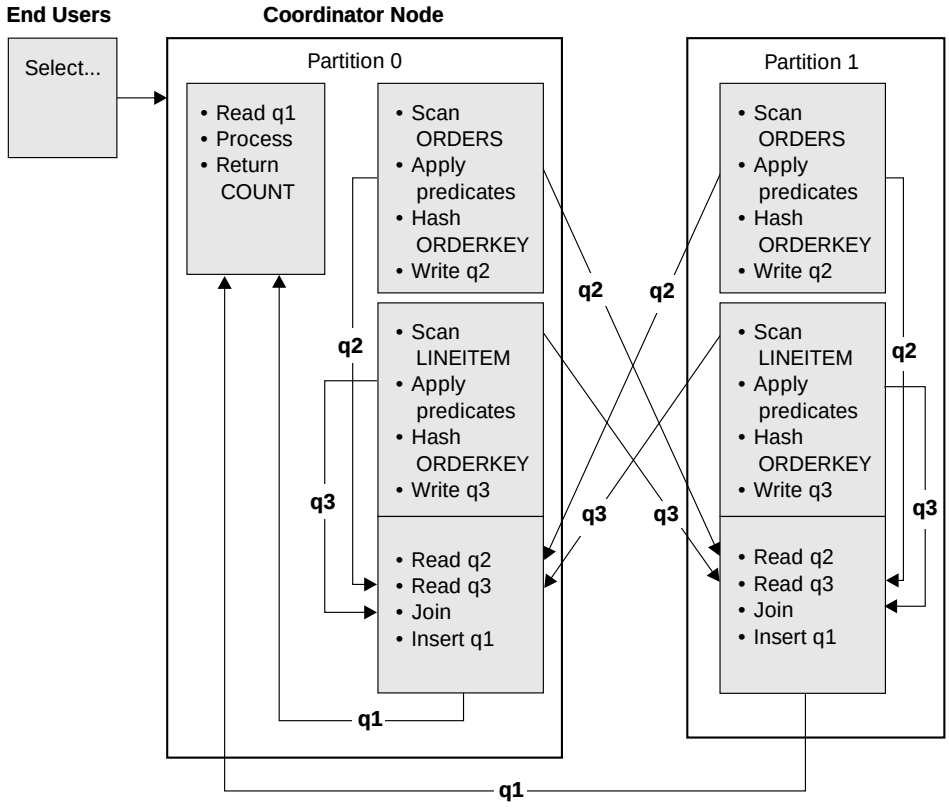


The LINEITEM table is partitioned on the ORDERKEY column.
The ORDERS table is partitioned on a different column.
The ORDERS table is hashed and sent to the correct LINEITEM table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 15. Directed Outer-Table Join Example

Directed Inner-Table and Outer-Table Joins

In the directed inner-table and outer-table join strategy, rows of both 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. The following figure provides an example. An example is shown in the following figure.

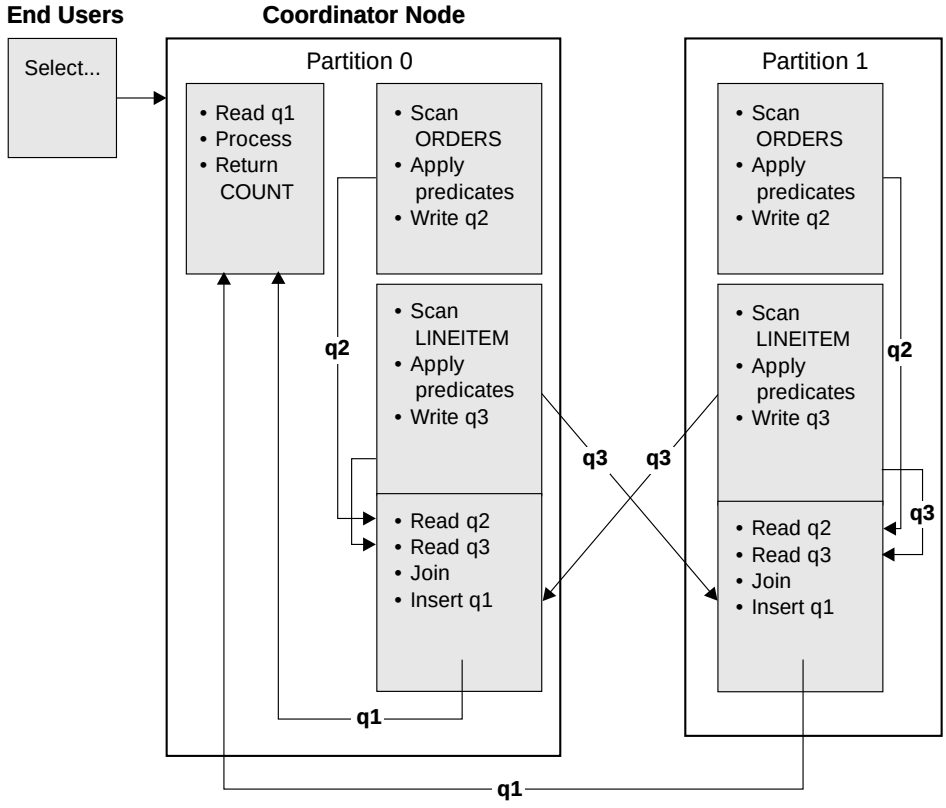


Neither table is partitioned on the ORDERKEY column.
Both tables are hashed and are sent to new database partitions where they are joined.
Both table queue q2 and q3 are directed.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY

Figure 16. Directed Inner-Table and Outer-Table Join Example

Broadcast Inner-Table Joins

In the broadcast inner-table join strategy, the inner table is broadcast to all the database partitions of the outer join table. The following figure provides an example.

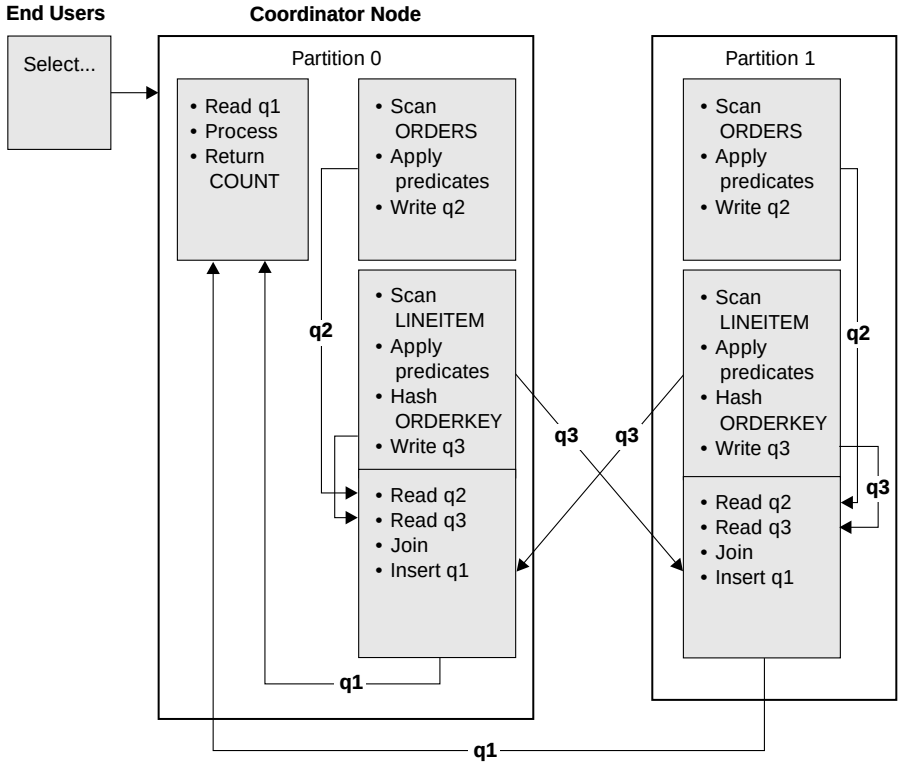


The LINEITEM table is sent to all database partitions that have the ORDERS table. Table queue q3 is broadcast to all database partitions of the outer table.

Figure 17. Broadcast Inner-Table Join Example

Directed Inner-Table Joins

With the directed inner-table join 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. The following figure provides an example.



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 18. Directed Inner-Table Join Example

Related concepts:

- "Joins" on page 187
- "Join methods" on page 188
- "Join strategies in partitioned databases" on page 196

Effects of sorting and grouping

When the optimizer chooses an access plan, it considers the performance impact of sorting data. Sorting occurs when no index satisfies the requested ordering of fetched rows. Sorting might also occur when the optimizer determines that a sort is less expensive than an index scan. The optimizer sort data in one of the following ways:

- Piping the results of the sort when the query is executed.
- Internal handling of the sort within the database manager.

Piped versus non-piped sorts

If the final sorted list of data can be read in a single sequential pass, the results can be *piped*. Piping is quicker than non-piped ways of communicating the results of the sort. The optimizer chooses to pipe the results of a sort whenever possible.

Whether or not a sort is piped, the sort time depends 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, in which each pass sorts a subset of the entire set of rows. Each sort pass is stored in a temporary table in the buffer pool. If there is not enough space in the buffer pool, pages from this temporary table might be written to disk. When all the sort passes are complete, these sorted subsets must be merged into a single sorted set of rows. If the sort is piped, the rows are handed directly to Relational Data Services as they are merged.

Group and sort pushdown operators

In some cases, the optimizer can choose to push down a sort or aggregation operation to Data Management Services 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 first passes this data to Relational Data Services, which then interfaces 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
```

Group operations in sorts

When sorting produces the order required for a GROUP BY operation, the optimizer can perform some or all of the GROUP BY aggregations 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.

An aggregation in sort requires as many as the following three stages of aggregation to ensure that proper results are returned.

1. The first stage of aggregation, partial aggregation, calculates the aggregate values until the sort heap is filled. In partial aggregation unaggregated data is taken in and partial aggregates are produced. If the sort heap is filled, the rest of the data spills to disk, including all of the partial aggregations that have been calculated in the current sort heap. After the sort heap is reset, new aggregations are started.
2. 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 uses existing partial aggregates to produce new partial aggregates. This stage does not always occur. It is used for both intra-partition and inter-partition parallelism. In intra-partition parallelism, the grouping is finished when a global grouping key is available. In inter-partition parallelism, this occurs when the grouping key is a subset of the partitioning key dividing groups across partitions, and thus requires repartitioning to complete the aggregation. A similar case exists in intra-partition parallelism when each agent finishes merging its spilled sort runs before reducing to a single agent to complete the aggregation.
3. The last stage of aggregation, final aggregation, uses all of the partial aggregates and produces final aggregates. This step always takes place in a GROUP BY operator. Sort cannot perform complete aggregation because they cannot guarantee that the sort will not split. Complete aggregation takes in unaggregated data and produces final aggregates. If partitioning does not prohibit its use, this method of aggregation is usually used to group data that is already in the correct order.

Optimization strategies

This section describes the particular strategies that the optimizer might use for intra-partition parallelism and multi-dimensional clustering (MDC) tables.

Optimization strategies for intra-partition parallelism

The optimizer can choose an access plan to execute a query in parallel within a single 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 specified when the SQL statement was compiled.

To parallelize an access plan, the optimizer divides 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 can send or receive data through table queues from subagents in other database partitions.

Intra-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 optimizer determines the scan unit (either a page or a row) and the scan granularity.

Parallel scans provide an even distribution of work among the subagents. The goal of a 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 strategies might cause data imbalance as the query executes. The optimizer chooses parallel strategies that maintain data balance among subagents.

Intra-partition parallel sort strategies

The optimizer can choose one of the following parallel sort strategies:

- **Round-robin sort**

  This is also known as a *redistribution sort*. This method uses shared memory efficiently redistribute the data as evenly as possible to all subagents. It uses a round-robin 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 to achieve 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 tables of a merge join are a partitioned sort, a subagent can use merge join to join the corresponding partitions and execute in parallel.

- **Replicated sort**

  This sort is used if each subagent requires all of the sort output. One sort is created and subagents are synchronized as rows are inserted into the sort. When the sort is completed, each subagent reads the entire sort. If the number of rows is small, this sort may be used to rebalance the data stream.

- **Shared sort**

  This sort is the same as a replicated sort, except the subagents open a parallel scan on the sorted result to distribute the data among the subagents in a way similar to the round-robin sort.

Intra-partition 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.

Intra-partition 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 and guarantees that all rows for a set of grouping columns are contained in one sort partition.
- If the stream needs to be replicated to balance processing, 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.

Intra-partition 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 or by replicating the data stream on the inner and outer tables of the join, or both. For example, a nested loop join can be parallelized if its outer stream is partitioned for a parallel scan and the inner stream is re-evaluated independently by each subagent. A merged join can be parallelized if its inner and outer streams are value-partitioned for partitioned sorts.

Related concepts:

- "Parallel processing for applications" on page 107
- "Optimization strategies for MDC tables" on page 209

Optimization strategies for MDC tables

If you create multi-dimensional clustering (MDC) tables, the performance of many queries might improve because the optimizer can apply additional optimization strategies. These strategies are primarily based on the improved efficiency of block indexes, but the advantage of clustering on more than one dimension also permits faster data retrieval.

Note: MDC table optimization strategies can also implement the performance advantages of intra-partition parallelism and inter-partition parallelism.

Consider the following specific advantages of MDC tables:

- Dimension block index lookups can identify the required portions of the table and quickly scan only the required blocks.
- Because block indexes are smaller than RID indexes, lookups are faster.
- Index ANDing and ORing can be performed at the block level and combined with RIDs.
- Data is guaranteed to be clustered on extents, which makes retrieval faster.

Consider the following simple example for an MDC table named **sales** with dimensions defined on the **region** and **month** columns:

```
SELECT * FROM SALES
    WHERE MONTH='March' AND REGION='SE'
```

For this query, the optimizer can perform a dimension block index lookup to find blocks in which the month of March and the SE region occur. Then it can quickly scan only the resulting blocks of the table to fetch the result set.

Related concepts:

- "Table and index management for MDC tables" on page 28

Automatic summary tables

Automatic summary tables (ASTs), a special kind of materialized query table, are a powerful way to improve response time for complex queries, especially queries that might require some of the following operations:

- Aggregated data over one or more dimensions
- Joins and aggregates data over a group of tables
- Data from a commonly accessed subset of data, that is, from a “hot” horizontal or vertical partition
- Repartitioned data from a table, or part of a table, in a partitioned database environment

Knowledge of summary tables is integrated into the SQL compiler. In the SQL compiler, the query rewrite phase and the optimizer match queries with summary tables and determine whether to substitute a summary table for a query that accesses the base tables. If summary tables are used, the EXPLAIN facility can provide information about which summary table was selected.

Because summary tables behave like regular tables in many ways, the same guidelines 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, the following example shows a multidimensional analysis query and how it takes advantage of summary tables.

In this example, assume a database scheme in which a data 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. This schema is classified as a multi-star because two large tables, the one containing transaction items and the other identifying the purchase transactions, are together are the hub of the star.

Three hierarchical dimensions that describe a transaction: product, location, and time. The product hierarchy is stored in two normalized tables representing the product group and the product line. The location hierarchy contains city, state, and country or region 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. Other tables in this schema represent account information for customers and customer information.

A summary table is created with the sum and count of sales for each level of the following hierarchies:

- Product
- Location
- Time, composed of year, month, day.

Many queries can be satisfied from this stored aggregate data. The following example shows how to create a summary table that 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 usually much smaller than the base fact tables. To control when the summary table is refreshed, specify the DEFERRED option as shown in the example.

Queries that can take advantage of such pre-computed sums would include the following:

- 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. Summary tables can reduce expensive joins, sorts, and aggregation of base data.

The following sample queries would obtain significant performance improvements because they can use the already computer results in the summary table.

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;
```

The larger the base tables are, the larger the improvements in response time can be because the summary table grows more slowly than the base table. Summary table can effectively eliminate overlapping work among queries by doing the computation once when the summary tables are built and refreshed and reusing their content for many queries.

Related concepts:

- "The Design Advisor" on page 242
- "Replicated materialized-query tables in partitioned databases" on page 194

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.

Federated database pushdown analysis

For queries in federated databases, the optimizer performs pushdown analysis to find out whether an operation can be performed at a remote data source. An operation might be a function, such as relational operator, system or user function, or an SQL operator, such as GROUP BY, ORDER BY, and so on.

Note: 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.

If functions cannot be pushed down to the remote data source, they can significantly impact query performance. Consider the effect of forcing a selective predicate to be evaluated locally instead of at the data source. Such evaluation could require DB2 to retrieve the entire table from the remote data source and then filter it locally against the predicate. Network constraints and large table size could cause performance to suffer.

Operators that are not pushed down can also significantly affect query performance. For example, having a GROUP BY operator aggregate remote data locally could also require DB2 to retrieve the entire table from the remote data source.

For example, assume that a nickname N1 references the data source table EMPLOYEE in a DB2 for OS/390® or z/OS data source. Also assume that the table has 10,000 rows, that one of the columns contains the last names of employees, and that one of the columns contains salaries. Consider the following statement:

```
SELECT LASTNAME, COUNT(*)  FROM N1
   WHERE LASTNAME  > 'B' AND SALARY > 50000
   GROUP BY LASTNAME;
```

Several possibilities are considered, depending on whether the collating sequences at DB2 and DB2 for OS/390 or z/OS are the same:

- If the collating sequences are the same, the query predicate can probably be pushed down to DB2 for OS/390 or z/OS. Filtering and grouping results at the data source is usually more efficient than copying the entire table to

DB2 and performing the operations locally. For the query above, the predicate and the GROUP BY operation can take place at the data source.

- If the collating sequence is not the same, the entire predicate cannot be evaluated at the data source. However, the optimizer might decide to pushdown the `SALARY > 50000` 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, the optimizer might decide that performing the `GROUP BY` operation locally at DB2 is the best (least cost) approach. The predicate is evaluated at the data source. This is an example of pushdown analysis combined with global optimization.

In general, the goal is to ensure that the optimizer evaluates functions and operators on data sources. Many factors affect whether a function or an SQL operator is evaluated at a remote data source. Factors to be evaluated are classified in the following three groups:

- Server characteristics
- Nickname characteristics
- Query characteristics

Server characteristics that affect pushdown opportunities

Certain data source-specific factors can affect pushdown opportunities. In general, these factors exist because of the rich SQL dialect supported by DB2. This dialect might offer more functionality than the SQL dialect supported by a server accessed by a 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 limit the number of items on the GROUP BY list or have 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 might have different SQL restrictions. For example, some data sources require parameter markers to bind 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 a value for a function, this function must be evaluated locally.

SQL Limits: Although DB2 might allow the use of larger integers than its remote data sources, values that exceed remote limits 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 whether NULL values are sorted as the highest or lowest value, or depend on the ordering. If NULL values are sorted at a data source differently from DB2, ORDER BY operations on a nullable expression cannot be remotely evaluated.

Collating Sequence: 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. If you configure a federated database to use the same collating sequence that a data source uses and then set the *collating_sequence* server option to 'Y', the optimizer can consider pushing down many query operations if improved performance results.

The following operations might be pushed down if collating sequences are the same:

- Comparisons of character or numeric data
- Character range comparison predicates
- Sorts

You might get unusual results, however, if the weighting of null characters is different between the federated database and the data source. Comparison statements might return unexpected results if you submit 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 assigns different weights to the characters.

To improve performance, the federated server allows sorts and comparisons to take place at data sources. For example, in DB2 UDB for OS/390 or z/OS, sorts defined by ORDER BY clauses are implemented by a collating sequence based on an EBCDIC code page. To use the federated server to retrieve DB2 for OS/390 or z/OS data sorted in accordance with ORDER BY clauses, configure the federated database so that it uses a predefined collating sequence based on the EBCDIC code page.

If the collating sequences of the federated database and the data source differ, DB2 retrieves the data to the federated database. Because users expect to see the query results ordered by the collating sequence defined for the federated server, by ordering the data locally the federated server ensures that this expectation is fulfilled. Submit your query in pass-through mode, or define the query in a data source view if you need to see the data ordered in the collating sequence of the data source.

Server Options: Several server options can affect pushdown opportunities. In particular, review your settings for *collating_sequence*, *varchar_no_trailing_blanks*, and *pushdown*.

DB2 Type Mapping and Function Mapping Factors: The default local data type mappings provided by DB2 are designed to provide sufficient buffer space for each data source data type, which avoids loss of data. Users can 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 might change the local data type to the DB2 DATE data type.

In the following three cases, DB2 can compensate for functions that a data source does not support:

- The function does not exist at the remote data source.
- The function exists, but 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 might return a different result if it is evaluated remotely. An example of this situation is the '>' (greater than) operator. For data sources with different collating sequences, the greater than operator might return different results than if it is evaluated locally by DB2.

Nickname characteristics that affect pushdown opportunities

The following nickname-specific factors 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. Use the default data type mappings to avoid 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 the longer column. This situation can affect the number of possibilities that the DB2 optimizer can evaluate in a joining sequence. 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 is 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: Use the SQL statement ALTER NICKNAME to add or change column options for nicknames.

Use the *varchar_no_trailing_blanks* option 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.

Use the *numeric_string* option to indicate whether the values in that column are always numbers without trailing blanks.

The table below describes these options.

Table 22. Column Options and Their Settings

Option	Valid Settings		Default Setting
numeric_string	'Y'	Yes, this column contains only strings of numeric data. IMPORTANT: If the column contains only numeric strings followed by trailing blanks, do not specify 'Y'.	'N'
	'N'	No, this column is not limited to strings of numeric data.	
	If you set 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 data. This option is useful when the collating sequence of a data source is different from DB2. Columns marked with this option are not excluded from local (data source) evaluation because of a different collating sequence.		

Table 22. Column Options and Their Settings (continued)

Option	Valid Settings	Default Setting
varchar_no_trailing_blanks	Specifies whether this data source uses non-blank padded VARCHAR comparison semantics. For variable-length character strings that contain no trailing blanks, non-blank-padded comparison semantics of some DBMSs return the same results as DB2 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 might have nicknames, including views. 'Y' This data source has non-blank-padded comparison semantics similar to DB2. 'N' This data source does not have the same non-blank-padded comparison semantics as DB2.	'N'

Query characteristics that affect pushdown opportunities

A query can reference an SQL operator that might involve nicknames from multiple data sources. The operation must take place at DB2 to combine the results from two referenced data sources that use one operator, such as a set operator (e.g. UNION). The operator cannot be evaluated at a remote data source directly.

Related concepts:

- "Guidelines for analyzing where a federated query is evaluated" on page 218

Guidelines for analyzing where a federated query is evaluated

DB2® provides two utilities to show where queries are evaluated:

- Visual explain. Start it with the **db2cc** 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 pushed down, you should see a RETURN operator. The RETURN operator is a standard DB2 operator. For a SELECT statement that selects data from a nickname, you also see a SHIP operator. The SHIP operator is unique to federated database operations. It changes the server property of the data flow and separates local operators from remote operators. The SELECT statement is generated using the SQL dialect supported by the data source. It can contain any valid query for that data source.

If an INSERT, DELETE, or UPDATE query can be entirely pushed down to the remote database, you might not see a SHIP statement in the access plan. All remotely executed INSERT, UPDATE, and DELETE statements are shown for the RETURN operator. However, if a query cannot be entirely pushed down, the SHIP operator shows which operations were performed remotely.

- 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

Consider the following key questions when you investigate ways to increase pushdown opportunities:

- 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 subquery 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.

- 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.
  - Does the GROUP BY operator clause contain a character expression? If it does, verify that the remote data source has the same case sensitivity as DB2.
- 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 or case sensitivity 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?

Related concepts:

- "Character-conversion guidelines" on page 105
- "Global analysis of federated database queries" on page 223
- "Remote SQL generation and global optimization in federated databases" on page 220
- "Federated query information" on page 629

Remote SQL generation and global optimization in federated databases

For a federated database query that uses relational nicknames, the access strategy might involve breaking down the original query into a set of remote query units and then combining the results. This generation of remote SQL helps produce a globally optimal access strategy for a query.

The optimizer uses the output of pushdown analysis to decide whether each operation is evaluated locally at DB2® or remotely at a data source. It bases its decision on the output of its cost model, which includes not only the cost of evaluating the operation but also the cost of transmitting the data or messages between DB2 and data sources.

Although the goal is to produce an optimized query, the following major factors affect the output from global optimization and thus affect query performance.

- Server characteristics
- Nickname characteristics

Server characteristics and options that affect global optimization

The following data source server factors can affect global optimization:

- Relative ratio of CPU speed

  Use the *cpu_ratio* server option to specify how fast or slow the data-source CPU speed is compared with the DB2 CPU. A low ratio indicates that the data-source computer CPU is faster than the DB2 computer CPU. If the ratio is low, the DB2 optimizer is more likely to consider pushing down CPU-intensive operations to the data source.

- 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. If the ratio is low, the DB2 optimizer considers pushing down I/O-intensive operations to the data source.

- Communication rate between DB2 and the data source

  Use the *comm_rate* server option to indicate network capacity. Low rates, which indicate 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 creates an access plan that requires minimal network traffic.

- Data source collating sequence

  Use the *collating_sequence* server option to specify whether 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.

- Remote plan hints

  Use the *plan_hints* server option to specify that plan hints should be generated or used at a data source. By default, DB2 does not send any plan hints to the data source.

  Plan hints are statement fragments that provide extra information for data-source optimizers. For some queries this information can improve 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 might 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 that affect global optimization

The following nickname-specific factors can affect global optimization.

Index considerations: To optimize queries, DB2 can use information about indexes at data sources. For this reason, it is important that the index information available to DB2 is current. The index information for nicknames is initially acquired when the nickname is created. 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 the DB2 optimizer to use. 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.

Consider creating index specifications in the following circumstances:

- DB2 cannot 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.

Before you issue CREATE INDEX statements against a nickname for a view, consider whether you need one. If the view is a simple SELECT on a table with an index, creating local indexes on the nickname to 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, such as a view created by joining two tables, query performance might suffer. For example, you create an index over a view that is a join of two tables, the optimizer might choose that view as the inner element in a nested-loop join. The query will have poor performance because the join is 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: System catalog statistics describe the overall size of nicknames and the range of values in associated columns. The optimizer uses these statistics when it calculates the least-cost path for processing queries that contain nicknames. Nickname statistics are stored in the same catalog views as table statistics.

Although DB2 can retrieve the statistical data stored at a data source, it cannot automatically detect updates to existing statistical data at data sources. Furthermore, DB2 cannot handle changes in object definition or structural changes, such as 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 and re-create it. Use this approach if structural information has changed.
- Manually update the statistics in the SYSSTAT.TABLES view. This approach requires fewer steps but it does not work if structural information has changed.

Related concepts:

- "Server options affecting federated databases" on page 115
- "Guidelines for analyzing where a federated query is evaluated" on page 218
- "Global analysis of federated database queries" on page 223

Global analysis of federated database queries

The following two utilities provided show global access plans:

- Visual Explain. Start it from the Control Center, or execute the *db2cc* command, which starts the Control Center. Use Visual Explain 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 SHIP or RETURN operator, depending on the type of the query. 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.

- SQL explain. Start it with the *db2expln* or *dynexpln* command. Use SQL explain to view the access plan strategy as text. Although SQL explain does not provide cost information, you can get the access plan generated by the remote optimizer for those data sources supported by the remote explain function.

Understanding DB2 optimization decisions

Consider the following optimization questions and key areas to investigate for performance improvements:

- 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.

  – 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.

  – 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 the CPU speed for DB2 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.

Related concepts:

- “Server options affecting federated databases” on page 115
- “Federated database pushdown analysis” on page 213
- “Guidelines for analyzing where a federated query is evaluated” on page 218
- “Remote SQL generation and global optimization in federated databases” on page 220
- “Federated query information” on page 629
- “Example five: federated database plan” on page 642

Chapter 7. SQL Explain facility

The Explain facility allows you to capture information about the access plan chosen by the optimizer as well as performance information that helps you tune queries.

SQL explain facility

The SQL compiler can capture information about the access plan and environment of static or dynamic SQL statements. The captured information helps you understand how individual SQL statements are executed so that you can tune the statements and your database manager configuration to improve performance.

You collect and use explain data for the following reasons:

- To understand how the database manager accesses tables and indexes to satisfy your query
- To evaluate your performance-tuning actions

  When you change some aspect of the database manager, the SQL statements, or the database, you should examine the explain data to find out how your action has changed performance./li>

The captured information includes:

- Sequence of operations to process the query
- Cost information
- Predicates and selectivity estimates for each predicate
- Statistics for all objects referenced in the SQL statement at the time that the explain information is captured

Before you can capture explain information, you create the relational tables in which the optimizer stores the explain information and you set the special registers that determine what kind of explain information is captured.

To display explain information, you can use either a command-line tool or Visual Explain. The tool that you use determines how you set the registry variables that determine what explain data is collected. For example, if you expect to use Visual Explain only, you need only capture snapshot information. If you expect to perform detailed analysis with one of the command-line utilities or with custom SQL statements against the explain tables, you should capture all explain information.

Related concepts:

- "Explain tools" on page 228
- "Guidelines for using explain information" on page 230
- "The explain tables and organization of explain information" on page 232
- "Guidelines for capturing explain information" on page 239
- "Guidelines for analyzing explain information" on page 241
- "SQL explain tools" on page 601

Tools for collecting and analyzing explain information

The Explain facility can be executed in several ways, depending on the kind of performance analysis that you want to perform. This section describes the tools that implement the Explain facility and the ways in which you might use the information collected.

Explain tools

DB2® provides a comprehensive explain facility that provides detailed information about the access plan that the optimizer chooses for an SQL statement. The tables that store explain data are accessible on all supported platforms and contain information for both static and dynamic SQL statements. Several tools or methods give you the flexibility you need to capture, display, and analyze explain information.

Detailed optimizer information that allows for in-depth analysis of an access plan is stored in explain tables separate from the actual access plan itself. Use one or more of the following methods of getting information from the explain tables:

- Use Visual Explain to view explain snapshot information

  Invoke Visual Explain from the Control Center to see a graphical display of a query access plan. You can analyze both static and dynamic SQL statements.

  Visual Explain allows 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.
- Use the *db2exfmt* tool to display explain information in preformatted output.
- Use the *db2expln* and *dynexpln* tools

  To see the access plan information available for one or more packages of static SQL statements, use the *db2expln* tool 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, use the explain facility.

The *db2expln* tool does provide a relatively compact and English-like overview of what operations will occur at run-time by examining the actual access plan generated.

- Write your own queries against the explain tables

  Writing your own queries allows for easy manipulation of the output and for comparison among different queries or for comparisons of the same query over time.

Note: The location of the command-line explain tools and others, such as *db2batch*, *dynexpln*, *db2vexp* , and *db2_all*, is in the *misc* subdirectory of the *sqllib* directory. If the tools are moved from this path, the command-line methods might not work.

The following table 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 23. Explain Facility Tools

Desired Characteristics	**Visual Explain**	**Explain tables**	**db2exfmt**	**db2expln**	**dynexpln**
GUI-interface	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		
Suited for analysis of multiple statements		Yes	Yes	Yes	Yes
Information accessible from within an application		Yes			

Table 23. Explain Facility Tools (continued)

Desired Characteristics	Visual Explain	Explain tables	db2exfmt	db2expln	dynexpln
Note:					
* Indirectly using db2expln; there are some limitations.					

Related concepts:

- Appendix D, "db2exfmt - Explain table-format tool" on page 645
- "db2expln syntax and parameters" on page 602
- "dynexpln" on page 610
- "Description of db2expln and dynexpln output" on page 610
- "Examples of db2expln and dynexpln output" on page 633

Guidelines for using explain information

You use explain information for the following two major purposes:

- To understand why application performance has changed
- To evaluate performance tuning efforts

Analysis of performance changes

To help you understand the reasons for changes in query performance, you need the before and after explain information. For this reason, perform the following steps:

- Capture explain information for the query before you make any changes and save the resulting explain tables, or you might save the output from the `db2exfmt` explain tool.
- Save or print the current catalog statistics if you do not want to, or cannot, access Visual Explain to view this information. You might also use the `db2look` productivity tool to help perform this task.
- Save or print the data definition language (DDL) statements, including those for CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE TABLESPACE.

The information that you collect in this way provides a reference point for future analysis. For dynamic SQL statements, you can 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. To analyze a performance change, you compare the information that you collected with information that you collect about the query and environment when you start your analysis.

As a simple example, your analysis might 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 perform one of these actions:

- Reorganize the index
- Collect new statistics for your table and indexes
- Gather explain information when rebinding your query.

After you perform one of the actions, examine the access plan again. If the index is used again, performance of the query might no longer be a problem. If the index is still not used or if performance is still a problem, perform a second action and examine the results. Repeat these steps until the problem is resolved.

Evaluation of performance tuning efforts

You can take a number of actions to help improve query performance, such as adjusting configuration parameters, adding containers, collecting fresh catalog statistics, and so on.

After you make 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 or automatic summary table (AST) based on the index guidelines, the explain data can help you determine whether the index or materialized query table is actually used as you expected.

Although the explain output provides information that allows 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.

Related concepts:

- Appendix D, “db2exfmt - Explain table-format tool” on page 645
- “SQL explain facility” on page 227
- “The explain tables and organization of explain information” on page 232
- “Guidelines for capturing explain information” on page 239
- “The Design Advisor” on page 242
- “Automatic summary tables” on page 210

Explain information collected

This section lists and describes each of the tables that store explain data, describes the kind of information that you can retrieve from the collected data, and provides guidelines for capturing information that is useful for your performance analysis.

The explain tables and organization of explain information

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. The explain information captured in one explain instance includes the SQL compilation environment as well as the access plan chosen to satisfy the SQL statement being compiled. For example, an explain instance might consist of any one of the following:

- All eligible SQL statements in one package for static SQL statements

  You can capture explain information for SELECT, SELECT INTO, UPDATE, INSERT, VALUES, VALUES INTO, and DELETE 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)

An explain instance represents the explain information for any of the following:

- 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)

Explain table information reflects the relationships between operators and data objects in the access plan. The following diagram shows the relationships between these tables.

Explain information is stored in the following tables:

Table 24. Relational tables that store explain data

Table Name	Description
EXPLAIN_ARGUMENT	Contains information about the unique characteristics for each individual operator, if any.
EXPLAIN_INSTANCE	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.

Table 24. Relational tables that store explain data (continued)

Table Name	Description
EXPLAIN_OBJECT	Identifies those data objects required by the access plan generated to satisfy the SQL statement.
EXPLAIN_OPERATOR	Contains all the operators needed to satisfy the SQL statement by the SQL compiler.
EXPLAIN_PREDICATE	Identifies the predicates that are applied by a specific operator.
EXPLAIN_STATEMENT	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 with the version used by the optimizer to choose an access plan. When an explain snapshot is requested, additional explain information is recorded to describe 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.
EXPLAIN_STREAM	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	Allows users to describe a workload to the database. Each row in the table represents an SQL statement in the workload and is described by an associated frequency. The `db2advis` tool uses this table to collect and store work and information.
ADVISE_INDEX	This table stores information about recommended indexes. The table can be populated by the SQL compiler, the `db2advis` utility or a user. This table is used in two ways: • To get recommended indexes. • To evaluate indexes based on input about proposed indexes.

Note: Not all of the tables above are created by default. To create them, run the EXPLAIN.DDL script found in the *misc* subdirectory of the *sqllib* subdirectory.

Explain tables might be common to more than one user. However, the explain tables can be defined for one user, and then aliases can be defined for each additional user using the same name to point to the defined tables. Each user sharing the common explain tables must have insert permission on those tables.

Related concepts:

• "SQL explain facility" on page 227

- "Explain information for data objects" on page 234
- "Explain information for instances" on page 235
- "Explain information for data operators" on page 234
- "SQL explain tools" on page 601

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 information about the object, such as the following:

- The creation time
- The last time that statistics were collected for the object
- An indication of whether or not the data in the object is ordered (only table or index objects)
- The number of columns in the object (only table or index objects)
- The estimated number of rows in the object (only table or index objects)
- 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 the index for this object and the number of levels in the tree
- The number of distinct full key values in the index for this object
- The total number of overflow records in the table

Related concepts:

- "The explain tables and organization of explain information" on page 232
- "Explain information for instances" on page 235
- "Explain information for data operators" on page 234
- "Guidelines for analyzing explain information" on page 241

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, such as a table scan, an index scan, a nested loop join, or a group-by operator.

In addition to showing the operators used in an access plan and information about each operator, explain information also shows 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 an invented relative unit of measure. Timerons are determined by the optimizer based on internal values such as statistics that change as the database is used. As a result, the timerons measure for a SQL statement are not guaranteed to be the same every time the estimated cost in timerons is determined.

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)

Related concepts:

- "The explain tables and organization of explain information" on page 232
- "Explain information for data objects" on page 234
- "Explain information for instances" on page 235
- "Guidelines for analyzing explain information" on page 241

Explain information for instances

Explain instance information is stored in the EXPLAIN_INSTANCE table. Additional specific information about each SQL statement in an instance is stored in the EXPLAIN_STATEMENT table.

Explain Instance Identification: Use this information to uniquely identify each explain instance and correlate the information for the SQL statements to a given invocation of the facility:

- The user who requested the explain information
- When the explain request began
- The name of the package that contains the explained SQL statement
- The schema of the package that contains the explained SQL statement
- The version of the package that contains the statement
- Whether snapshot information was collected

Environmental Settings: Information about the database manager environment in which the SQL compiler optimized your queries is captured. The environmental information includes the following:

- The version and release number for the level of DB2®
- The degree of parallelism for which the query was compiled

  The CURRENT DEGREE special register, the DEGREE bind option, the SET RUNTIME DEGREE API, and the *dft_degree* configuration parameter determine the degree of parallelism for which a particular query is compiled.
- Whether the SQL statement is dynamic or static
- The query optimization class used to compile the query
- The type of row blocking for cursors specified when compiling the query
- The isolation level in which the query runs
- The values of various configuration parameters when the query was compiled. The following parameters are recorded when an explain snapshot is taken:
  - Sort Heap Size (*sortheap*)
  - Average Number of Active Applications (*avg_appls*)
  - Database Heap (*dbheap*)
  - Maximum Storage for Lock List (*locklist*)
  - Maximum Percent of Lock List Before Escalation (*maxlocks*)
  - CPU Speed (*cpuspeed*)
  - Communications Bandwidth (*comm_bandwidth*)

SQL Statement Identification: More than one SQL statement might have been explained for each explain instance. In addition to information that uniquely identifies the explain instance, the following information helps identify individual SQL statements:

- 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

The QUERYTAG and QUERYNO fields in the EXPLAIN_STATEMENT table contain identifiers that are set 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". In this case, the QUERYNO value defaults to a number that is incremented by one or more for each statement. For all other dynamic explain SQL statements, which are not from CLP, CLI, or do not use the EXPLAIN SQL statement, QUERYTAG is set to blanks and QUERYNO is always "1".

Cost Estimation: For each explained statement, the optimizer records an estimate of the relative cost of executing the chosen access plan. This cost is stated in an invented relative unit of measure called a *timeron*. No estimate of elapsed times is provided, for the following reasons:

- The SQL optimizer does not estimate elapsed time but only resource consumption.
- The optimizer does not model all factors that can affect elapsed time. It ignores factors that do not affect the efficiency of the access plan. A number of runtime factors affect the elapsed time, 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: Two versions of the text of the SQL statement are recorded for each statement explained. One version is the code that the SQL compiler receives from the application. The other version is reverse-translated from the internal compiler representation of the query. Although this translation looks similar to other SQL statements, it 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 only to allow you to understand the SQL context in which the SQL optimizer chose the access plan. To understand how the SQL compiler has rewritten your query for better optimization, compare the user-written statement text to the internal representation of the SQL statement. The rewritten statement also shows you other elements in the environment affecting your statement, such as triggers and constraints. Some keywords used by this "optimized" text are:

$C*n*	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.T*n* The name of a derived table, where *n* represents an integer value.

$INTERNAL_FUNC$ The tag indicates the presence of a function used by the SQL compiler for the explained query but not available for general use.

$INTERNAL_PRED$ The tag indicates the presence of a predicate added by the SQL compiler during compilation of the explained query but not available for general use. An internal predicate is used by the compiler to satisfy additional context added to the original SQL statement because 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 appears at the start of the text when additional triggers or constraints have been added into the original SQL statement. A list of the names of any triggers or constraints affecting the compilation and resolution of the SQL statement appears after this prefix.

Related concepts:

- "The explain tables and organization of explain information" on page 232
- "Explain information for data objects" on page 234
- "Explain information for data operators" on page 234
- "Guidelines for analyzing explain information" on page 241

Related reference:

- "Communications Bandwidth configuration parameter - comm_bandwidth" on page 513
- "Sort Heap Size configuration parameter - sortheap" on page 405
- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428

- "Database Heap configuration parameter - dbheap" on page 392
- "CPU Speed configuration parameter - cpuspeed" on page 513
- "Average Number of Active Applications configuration parameter - avg_appls" on page 440
- "Default Degree configuration parameter - dft_degree" on page 491

Guidelines for capturing explain information

Explain data is captured if you request it when an SQL statement is compiled. Consider how you expect to use the captured information when you request explain data.

Notes:

1. If incremental bind SQL statements are compiled at run-time, data is placed in the explain tables at run-time and not bind-time. For these statements, the explain table qualifier and authorization ID inserted is that of the package owner and not that of the user running the package.
2. Explain information is captured only when the SQL statement is compiled. After the initial compilation, dynamic SQL statements are recompiled only when a change to the environment requires it. If you issue the same PREPARE statement for the same SQL statement, the SQL statement is compiled and explain data is captured only the first time you issue the PREPARE statement if the environment does not change.

Capturing information in the explain tables

- **Static or incremental bind SQL statements**:

  Specify either EXPLAIN ALL or EXPLAIN YES options on the BIND or the PREP commands or include a static EXPLAIN SQL statement in the source program.

- **Dynamic SQL statements**:

  Explain table information is captured in any of the following cases:

  – The CURRENT EXPLAIN MODE special register is set to:
    - YES: The SQL compiler captures explain data and executes the SQL statement.
    - EXPLAIN: The SQL compiler captures explain data, but does not execute the SQL statement.
    - RECOMMEND INDEXES: The SQL compiler captures explain data and the recommended indexes are placed in the ADVISE_INDEX table, but the SQL statement is not executed.
    - EVALUATE INDEXES: The SQL compiler uses indexes placed by the user in the ADVISE_INDEX table for evaluation. In EVALUATE INDEXES mode, all dynamic statements 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. To find out if proposed indexes are useful, review the EXPLAIN results.

– The EXPLAIN ALL option has been specified on the BIND or PREP command. The SQL compiler captures explain data for dynamic SQL at run-time, even if the CURRENT EXPLAIN MODE special register is set to NO. The SQL statement also executes and returns the results of the query.

Capturing explain snapshot information

When an explain snapshot is requested, explain 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. This information includes information about data objects and data operators.

Explain snapshot data is captured when an SQL statement is compiled and explain data has been requested, as follows:

- **Static or incremental bind SQL statements**:

  An explain snapshot is captured when either EXPLSNAP ALL or EXPLSNAP YES clauses are specified on the BIND or the PREP commands or when the source program includes a static EXPLAIN SQL statement that uses a FOR SNAPSHOT or a WITH SNAPSHOT clause.

- **Dynamic SQL statements**:

  An explain snapshot is captured in any of the following cases:

  – You issue an EXPLAIN SQL statement with a FOR SNAPSHOT or a WITH SNAPSHOT clause. With the FOR SNAPSHOT clause, only explain snapshot information is captured. With the WITH SNAPSHOT clause, all explain information is captured in addition snapshot information.

  – The CURRENT EXPLAIN SNAPSHOT special register is set to:

    - YES: The SQL compiler captures snapshot explain data and executes the SQL statement.
    - EXPLAIN: The SQL compiler captures snapshot explain data, but does not execute the SQL statement.

  – You specify the EXPLSNAP ALL option on the BIND or PREP command. The SQL compiler captures snapshot explain data at run-time, even if the setting of the CURRENT EXPLAIN SNAPSHOT special register is NO. It also executes the SQL statement.

Related concepts:

- "SQL explain facility" on page 227
- "Guidelines for using explain information" on page 230
- "The explain tables and organization of explain information" on page 232
- "The Design Advisor" on page 242
- "Guidelines for analyzing explain information" on page 241
- "SQL explain tools" on page 601

Guidelines for analyzing explain information

Although the primary use of explain information is analysis of the access paths for SELECT statements, there are a number of ways in which analyzing the explain data can help you to tune your queries and environment. Consider the following kind of analysis:

- **Index use**

  The proper indexes can significantly benefit 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 might be used instead of an existing index or no index at all. After you create a new index, use the RUNSTATS command to collect statistics for that index and recompile the 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 the table to cluster its data according to that index, use the RUNSTATS command to collect statistics for both index and table, and then recompile the query. To determine whether reorganizing table has improved the access plan, re-examine the explain output for the recompiled query.

- **Access type**

  Analyze the explain output and look for types of access to the data that are not usually 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 users usually look at only a few screens of the output data, you might specify 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. If you specify this option, 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.

- **Join methods**

  If a query joins two tables, check the type of join being used. Joins that involve more rows, such as those in decision-support queries, usually run faster with a merge join. Joins that involve 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 might change how these typical joins work.

Related concepts:

- "SQL explain facility" on page 227
- "SQL explain tools" on page 601
- "Description of db2expln and dynexpln output" on page 610

The Design Advisor

The Design Advisor can help you design and define suitable indexes for your data by helping you:

- Find the best indexes for a problem query.
- Find the best indexes for the set of queries that define a workload, subject to resource limits that are optionally applied.

- Test an index or materialized query table on a workload without having to create the index or materialized query table.

The Design Advisor uses the following terms and concepts:

Workload: A *workload* is a set of SQL statements which the database manager has to process during a given period of time. The SQL statements can include SELECT, INSERT, UPDATE, and DELETE statements. For example, during one month 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.

Virtual index: *Virtual indexes* are indexes that do not exist in the current database schema. These indexes might be either recommendations that the advise facility has made, or indexes that you want the advise facility to evaluate. These indexes might also be indexes the advise facility considers as part of the process and then discards because they are not recommended. Virtual index information is defined and manipulated in the ADVISE_INDEX table. Virtual materialized query tables are considered in the same way as virtual indexes.

The Design Advisor uses a workload and statistics from the database to generate recommended indexes. It uses the following EXPLAIN tables:

- ADVISE_WORKLOAD

  You add data to this table to describe the workload. Each row represents an SQL statement and is described by an associated frequency. Each workload is identified by the string in the WORKLOAD_NAME column in the row. All SQL statements in the same workload use the same WORKLOAD_NAME. The Design Advisor and the *db2advis* tool retrieve and store workload information in this table.

- ADVISE_INDEX

  This table stores information about recommended indexes. Information is written to this table by the SQL compiler, by the Design Advisor, by the `db2advis` tool, or by you.

  The table is used in two ways:

  – To get recommended indexes from the Design Advisor

  – To evaluate indexes

Note: These tables are created when you run the EXPLAIN.DDL script that creates the other relational tables for the explain facility.

To use the DB2 advisor you create input that describes the workload, start the DB2 advisor, and evaluate outputs. Some special cases should be considered.

Create the input for the DB2 advisor in one of the following ways:

- Capture a workload.

  Use one of the following ways to create the SQL to be evaluated:

  – Use the monitor to get dynamic SQL.
  – Use the SYSSTMT catalog view to get static SQL.
  – Add statements and frequencies by cutting and pasting the values into the ADVISE_INDEX table.
- Modify the workload frequencies to increase or decrease the importance of queries.
- Determine the constraints, if any, on the data.

Start the DB2 advisor in one of the follow ways:

- Use the Control Center.

  This is the recommended method. Extensive help pages can answer your questions. To help you construct a workload, the wizard can look for recently executed SQL statements, or look through the recently used packages, or let you add SQL statements manually.
- Use the command-line processor.

  On the command line enter `db2advis`. When `db2advis` starts, it reads a workload from one of three locations:

  – The command line
  – The statements in a text file
  – The ADVISE_WORKLOAD table after you have inserted rows with the proposed workload (SQL and frequency).

  The `db2advis` 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 is sent to your terminal screen, the ADVISE_INDEX table, and an output file, if desired.

  For example, you might ask 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 recommended 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.
```

- Use self-directed methods involving the EXPLAIN modes and PREP command options.

  For example, if the CURRENT EXPLAIN MODE special register is set to RECOMMEND INDEXES, the SQL compiler captures explain data and the recommended indexes are placed in the ADVISE_INDEX table, but the SQL statement is not executed.

  If the CURRENT EXPLAIN MODE special register is set to EVALUATE INDEXES, the SQL compiler uses indexes that the user defines in the ADVISE_INDEX table. For each index to be evaluated, the user inserts a new row in the ADVISE_INDEX table. Each row contains the index name, table name, and the columns names that make up the index being evaluated. Then when the CURRENT EXPLAIN MODE special register is set to EVALUATE INDEXES, the SQL compiler scans the ADVISE_INDEX table for rows in which 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 either chooses to use the virtual indexes or ignores them, depending on whether a particular index improves performance. To see if the indexes that you propose are used by the compiler, review the EXPLAIN results. To improve access, you might create the indexes that the compiler indicates that it would use.

- Using the Call Level Interface (CLI).

  If you are using this interface to write applications, you can also use the DB2 advisor.

Use DB2 advisor results in the following ways:

- Interpret the output from the DB2 advisor.

  To see what indexes were recommended by the advise facility, use the following query:

  ```
  SELECT CAST(CREATION_TEXT as CHAR(200))
     FROM ADVISE_INDEX
  ```

- Apply the recommendations of the DB2 advisor.
- Know when to drop an index.

To get better recommendations for a specific query, you might advise that query by itself. You can use the Design Advisor to recommend indexes for a single query by building a workload which contains only that query.

Because the Event Monitor can collect dynamic SQL execution, you can collect a sample workload from Event Monitor output. Then these statements can be fed back to the DB2 advisor.

Related concepts:

- "Advantages and disadvantages of indexes" on page 294
- "Index planning tips" on page 296
- "Automatic summary tables" on page 210

Related reference:

- "db2advis - DB2 Index Advisor Command" in the *Command Reference*

Part 3. Tuning and configuring your system

Chapter 8. Operational performance

This chapter provides information about factors that affect the performance of SQL queries at runtime. You might also refer to planning information about physical database design considerations, particularly for the advantages of partitioning, multi-dimensional clustering (MDC) tables, and similar features.

Memory usage

This section describes how the database manager uses memory and lists the parameters that control the database manager and database use of memory.

Organization of memory use

Understanding how DB2® organizes memory helps you tune memory use for good performance. Many configuration parameters affect memory usage. 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. While the database server is running, you can increase or decrease the size of memory areas inside the database shared memory.

A system administrator needs to consider overall balance of memory usage on the system. Different applications running on the operating system might use memory in different ways. For example, although some applications may use the operating-system cache, the database manager uses its own buffer pool for data caching instead of the operating-system cache.

The figure below shows different portions of memory that the database manager allocates for various uses.

Note: This figure does not show how memory is used in an Enterprise Server Edition environment, which comprises multiple logical nodes. In such an environment, each node contains a Database Manager Shared Memory set.

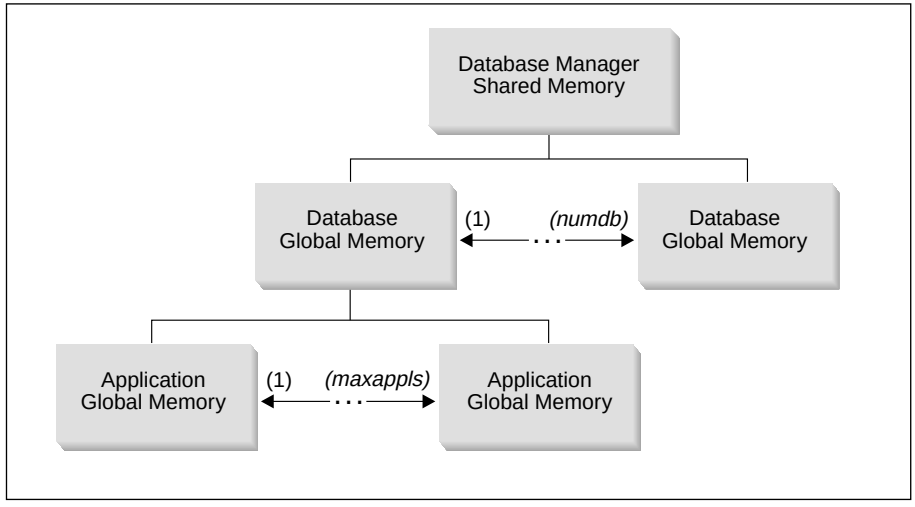


Figure 19. Types of memory used by the Database Manager

Memory is allocated for each instance of the database manager when the following events occur:

- **When the database manager is started (`db2start`):** Database manager global shared memory is allocated and remains allocated until the database manager is stopped (`db2stop`). This area contains information that the database manager uses 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:** Database global memory is allocated. Database global memory is used across all applications that might connect to the database. The size of the database global memory is specified by the *database_memory* configuration parameter. You can specify more memory than is needed initially so that the additional memory can be dynamically distributed later. Although the total amount of database global memory cannot be increased or decreased while the database is active, memory for areas contained in database global memory can be adjusted. Such areas include the buffer pools, the lock list, the database heap and utility heap, and the package cache, and the catalog cache.
- **When an application connects to a database:** In a partitioned database environment, in a non-partitioned database with the database manager intra-partition parallelism configuration parameter (*intra_parallel*) enabled, or in an environment in which the connection concentrator is enabled, multiple applications can be assigned to *application groups* to share memory. Each application group has its own allocation of shared memory. In the

application-group shared memory, each application has its own application control heap but uses the share heap of the application group.

The following three database configuration parameters determine the size of the application group memory:

- The *appgroup_mem_sz* parameter, which specifies the size of the shared memory for the application group
- The *groupheap_ratio* parameter, which specifies the percent of the application-group shared memory allowed for the shared heap
- The *app_ctl_heap_sz* parameter, which specifies the size of the control heap for each application in the group.

The performance advantage of grouping application memory use is improved cache and memory-use efficiency.

Some elements of application global memory can also be resized dynamically.

- **When an agent is created:** This event is not shown in the figure. Agent private memory is allocated for an agent when the agent is assigned as the result of a connect request or a new SQL request in a parallel environment, Agent private memory is allocated for the agent and contains memory allocations that is used only by this specific agent, such as the sort heap and the application heap.

  When a database is already in use by one application, only agent private memory and application global shared memory is allocated for subsequent connecting applications.

The figure also lists the following configuration parameter settings, which limit the amount of memory that is allocated for each specific purposes. Note that in a partitioned database environment, this memory is allocated on each database partition.

- *numdb*

  This parameter specifies the maximum number of concurrent active databases that different applications can use. Because each database has its own global memory area, the amount of memory that might be allocated increases if you increase the value of this parameter. When you set *numdb* to *automatic*, you can create any number of databases and memory consumption grows accordingly.

- *maxappls*

  This parameter specifies the maximum number of applications that can simultaneously connect to a single database. It affects the amount of memory that might be allocated for agent private memory and application global memory for that database. Note that this parameter can be set

differently for every database. When you set *maxappls* to *automatic*, you can create any number of databases and memory consumption grows accordingly.

- *maxagents* and *max_coordagents* for parallel processing

  These parameters are not shown in the figure. They limit the number of database manager agents that can exist simultaneously across all active databases in an instance. Together with *maxappls*, these parameters limit the amount of memory allocated for agent private memory and application global memory.

Related concepts:

- "Database manager shared memory" on page 254
- "The FCM buffer pool and memory requirements" on page 256
- "Global memory and parameters that control it" on page 258
- "Guidelines for tuning parameters that affect memory usage" on page 261
- "Memory management" on page 44

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.

The following figure shows how memory is used to support applications. The configuration parameters shown allow you to control the size of this memory, by limiting the number and size of memory segments, which are portions of logical memory.

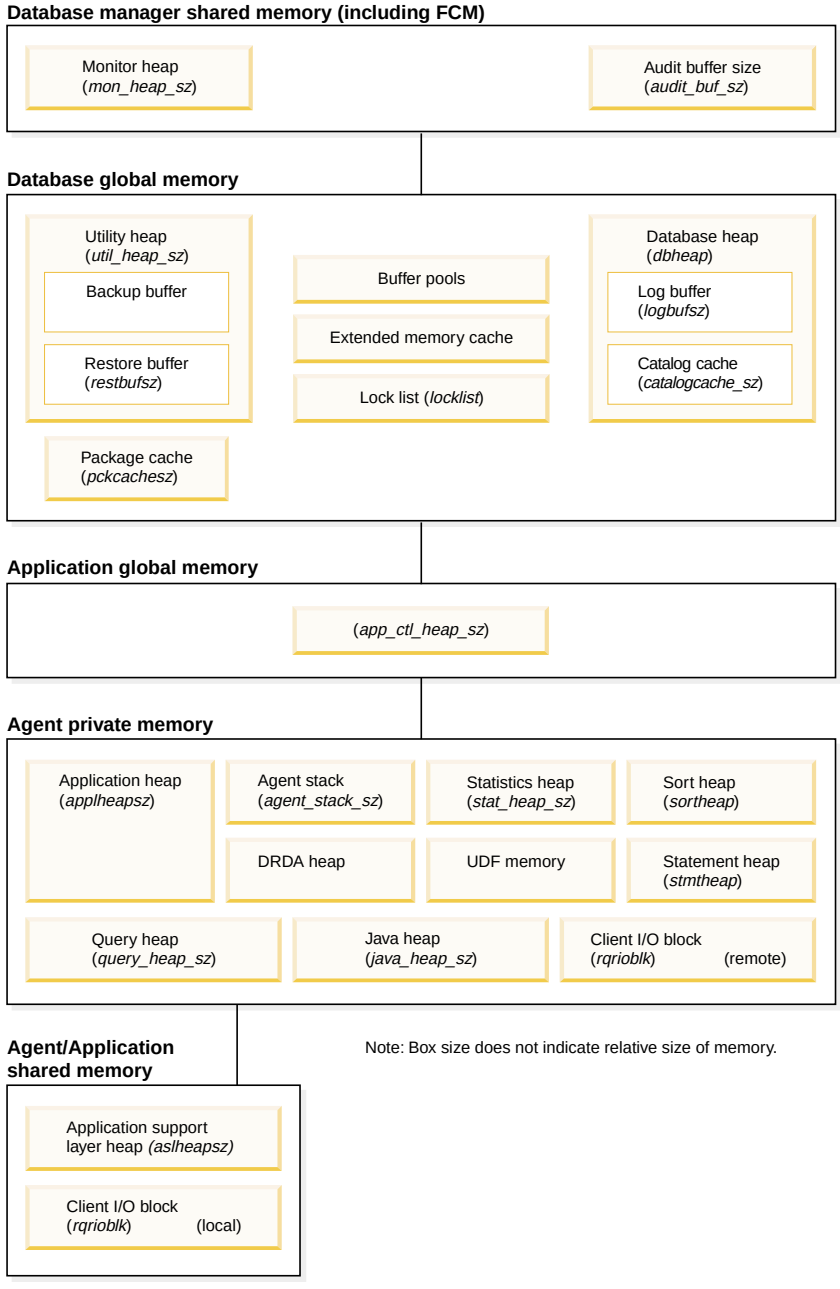


Figure 20. How memory is used by the database manager

You can predict and control the size of this space by reviewing information about database agents. Agents running on behalf of applications require substantial memory space, especially if the value of *maxagents* is not appropriate.

For partitioned database systems, the fast communications manager (FCM) requires substantial memory space, especially if the value of *fcm_num_buffers* is large. In addition, 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.

Related concepts:

- "Organization of memory use" on page 251
- "Global memory and parameters that control it" on page 258
- "Buffer-pool management" on page 264
- "Guidelines for tuning parameters that affect memory usage" on page 261
- "Memory management" on page 44

The FCM buffer pool and memory requirements

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 below.

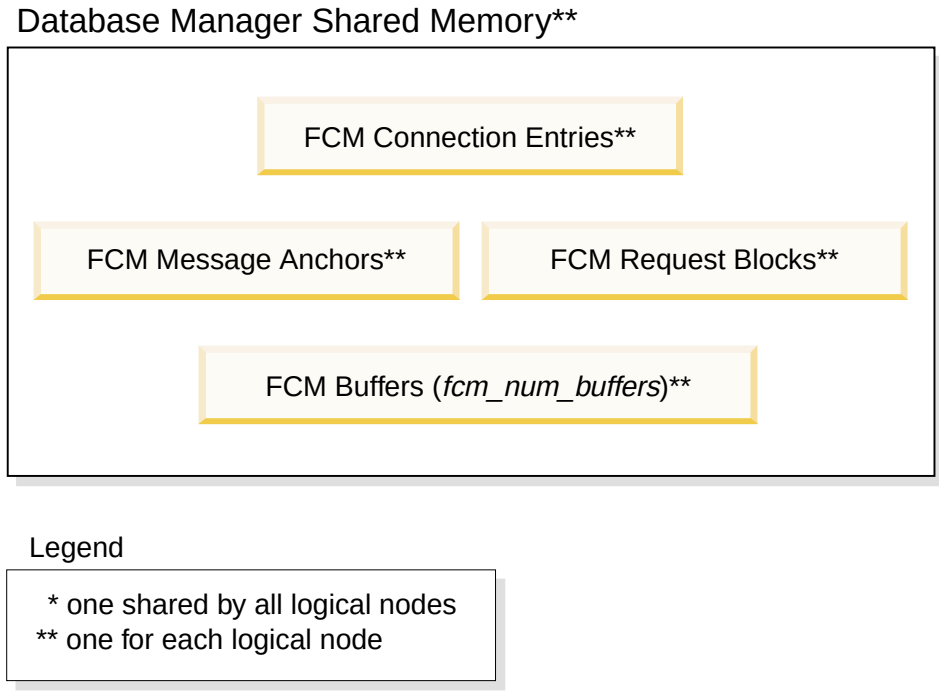


Figure 21. FCM buffer pool when multiple logical nodes are not used

If you have a partitioned database system that uses multiple logical nodes, the Database Manager Shared Memory and FCM Buffer Pool are as shown below.

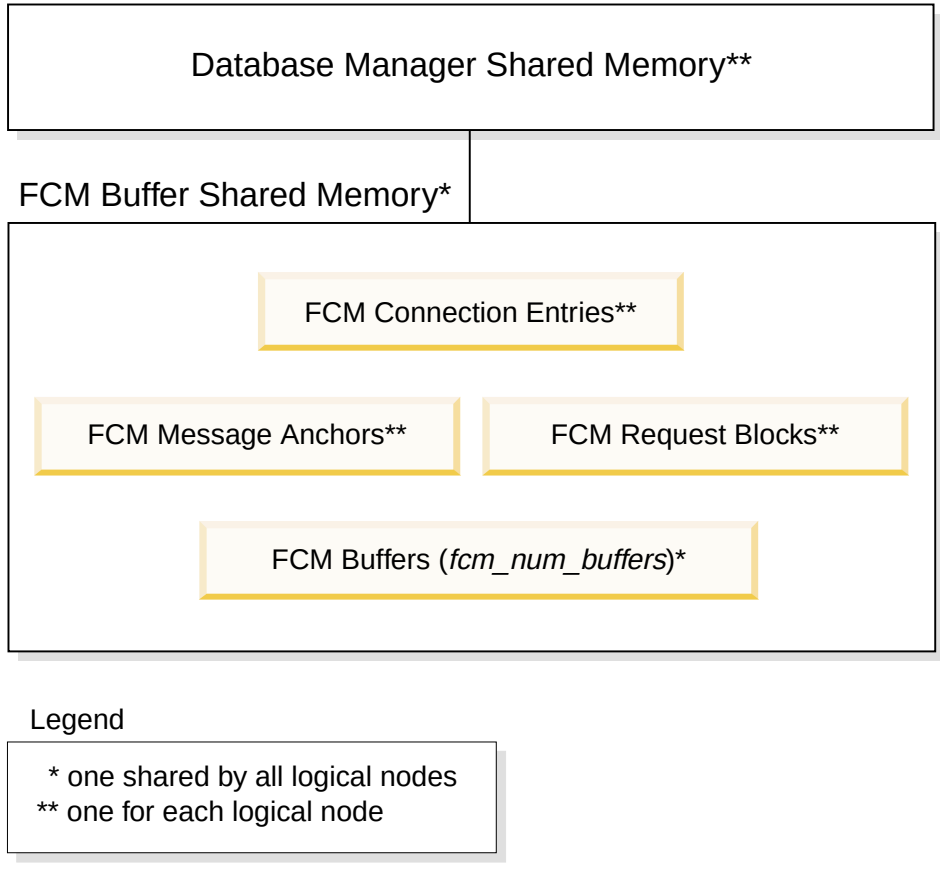


Figure 22. FCM buffer pool when multiple logical nodes are used

For configuring the fast communications manager (FCM), start with the default value for the number of FCM Buffers (*fcm_num_buffers*). For more information about FCM on AIX® platforms, refer to the description of the DB2_FORCE_FCP_BP registry variable.

To tune this parameter, use the database system monitor to monitor the low water mark for the free buffers.

Related concepts:

- "Database manager shared memory" on page 254

Global memory and parameters that control it

Database manager shared memory is made up of the following components:

Database Global Memory

Database Global Memory is affected by the following configuration parameters:

- The *database_memory* parameter provides a lower bound for the size of the database global memory.
- The following parameters or factors specify the maximum size of memory segments:
  - The size of the buffer pools.
  - Maximum Storage for Lock List (*locklist*)
  - Database Heap (*dbheap*)
  - Utility Heap Size (*util_heap_sz*)
  - Extended Storage Memory Segment Size (*estore_seg_sz*)
  - Number of Extended Storage Memory Segments (*num_estore_segs*)
  - Package Cache Size (*pckcachesz*)

Application Global Memory

Application Global Memory is affected by the Application Control Heap Size (*app_ctl_heap_sz*) configuration parameter.

For parallel systems, space is also required for the application control heap, which is shared between the agents that are working for the same application at one database partition. The heap is allocated when a connection is requested by the first agent to receive a request from the application. The agent can be either a coordinating agent or a subagent.

Agent Private Memory

- The number of memory segments is limited by the lower of:
  - The total of the *maxappls* configuration parameter for all active databases, that specifies the maximum number of active applications permitted.
  - The value of the *maxagents* configuration parameter, which specifies the maximum number of agents permitted.
- The maximum size of memory segments is determined by the values of the following parameters:
  - Application Heap Size (*applheapsz*)
  - Sort Heap Size (*sortheap*)
  - Statement Heap Size (*stmtheap*)
  - Statistics Heap Size (*stat_heap_sz*)
  - Query Heap Size (*query_heap_sz*)
  - Agent Stack Size (*agent_stack_sz*)

Agent/Application Shared Memory

- The total number of agent/application shared memory segments for local clients is limited by the lower of the following database configuration parameters:
  - The total of *maxappls* for all active databases
  - The value of *maxagents* , or *max_coordagents* for parallel systems.
- Agent/Application Shared Memory is also affected by the following database configuration parameters:
  - The Application Support Layer Heap Size (*aslheapsz*)parameter
  - The Client I/O Block Size (*rqrioblk*) parameter

Related reference:

- "Extended Storage Memory Segment Size configuration parameter - estore_seg_sz" on page 437
- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Number of Extended Storage Memory Segments configuration parameter - num_estore_segs" on page 437
- "Sort Heap Size configuration parameter - sortheap" on page 405
- "Maximum Number of Agents configuration parameter - maxagents" on page 444
- "Application Support Layer Heap Size configuration parameter - aslheapsz" on page 417
- "Application Heap Size configuration parameter - applheapsz" on page 410
- "Package Cache Size configuration parameter - pckcachesz" on page 399
- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Client I/O Block Size configuration parameter - rqrioblk" on page 420
- "Database Heap configuration parameter - dbheap" on page 392
- "Statement Heap Size configuration parameter - stmtheap" on page 409
- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438
- "Agent Stack Size configuration parameter - agent_stack_sz" on page 412
- "Query Heap Size configuration parameter - query_heap_sz" on page 411
- "Utility Heap Size configuration parameter - util_heap_sz" on page 396
- "Statistics Heap Size configuration parameter - stat_heap_sz" on page 410
- "Database Shared Memory Size configuration parameter - database_memory" on page 391

Guidelines for tuning parameters that affect memory usage

The first rule for setting memory-allocation parameters is never to set them at their highest values unless such a value has been carefully justified. This rule applies even to systems with the maximum amount of memory. Many parameters that affect memory can allow the database manager easily and quickly to take up all of the available memory on a computer. In addition, managing large amounts of memory requires additional work on the part of the database manager and thus incurs more overhead.

Some UNIX® operating systems allocate swap space when a process allocates memory and not when a process is paged out to swap space. For these systems, make sure that you provide as much paging space as total shared memory space.

For most configuration parameters, memory is only committed as it is required. These parameters determine the maximum size of a particular memory heap. In the following cases, however, the full amount of memory specified by the parameter is allocated:

- Sort Heap Threshold (*sheapthres*)
- Maximum Storage for Lock List (*locklist*)
- Application Support Layer Heap Size (*aslheapsz*)
- Number of FCM Buffers (*fcm_num_buffers*)
- Catalog cache size (*catalogcache_sz*)
- Package cache size (*pckcachesz*)
- Utility heap size (*util_heap_sz*)
- Database heap size (*dbheap*)

Note: To reduce the need for increasing the *dbheap* when buffer pools increase in size, nearly all bufferpool memory, including page descriptors, buffer-pool descriptors, and the hash tables comes out of the database shared-memory set and is sized automatically.

Note: To change the size of a buffer pool, use the DDL statement, ALTER BUFFERPOOL. Changes to the buffer pool take effect the next time the database is started. The current attributes of the buffer pool are in effect until then.

Parameters that affect application-group memory use

The parameters that affect application-group use of memory apply only to partitioned databases, databases for which intra-parallel processing is enabled,

and databases for which the connection concentrator is enabled. The following parameters determine how applications in application groups use their shared memory:

- The *appgroup_mem_sz* parameter specifies the size of the shared memory for the application group.

  Setting the *appgroup_mem_sz* configuration parameter too high has an adverse effect. Because the all applications in the application group share the caches in the application-group heap, having too many applications will increase cache contention. On the other hand, if each application group contains few applications, the effect of the cache is also limited.
- The *groupheap_ratio* parameter specifies the percent of memory allowed for the shared heap.

  Setting *groupheap_ratio* too low limits the size of caches. Setting the *groupheap_ratio* too high causes the application control heap to be too small and might cause SQL error `SQL0973`, which warns you that you are running out of application control-heap memory at run time.
- The *app_ctl_heap_sz* parameter specifies the size of the control heap for each application in the group.

Accept the default setting for these parameters when you configure your database server. Adjust the settings only if performance suffers. For example, set *appgroup_mem_sz* to control the number of applications in each application group. As a rule of thumb, consider that 10 is too small and 100 is too many. The default is probably appropriate. Then run an average workload and use the Health Center utility from the Control Center or the system monitor to collect information about the hit ratios for the catalog cache, the package cache, and the shared workspace.

- If many `sql0973` errors occur, the *groupheap_ratio* setting is too high.
- If the memory tracker shows that the average and maximum usage of the application control heap is far below `app_ctl_heap_sz * (100 - groupheap_ratio)/ 100`, reduce the value of the *app_ctl_heap_sz* configuration parameter.
- If the cache usage indicates that the caches are reaching their limit, increase the value of the *group_heap_ratio* configuration parameter or reduce the number of of applications in the application group.

Notes:

- Benchmark tests provide the best information about setting appropriate values for memory parameters. In benchmarking, 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 is graphed, the point at which the curve begins to

plateau or decline indicates the point at which additional allocation provides no additional value to the application and is therefore simply wasting memory.

- The upper limits of memory allocation for several parameters may be beyond the memory capabilities of existing hardware and operating systems. These limits allow for future growth.
- For valid parameter ranges, refer to the detailed information about each parameter.

Related concepts:

- "Organization of memory use" on page 251
- "Database manager shared memory" on page 254
- "Global memory and parameters that control it" on page 258

Related reference:

- "Application Control Heap Size configuration parameter - app_ctl_heap_sz" on page 403
- "Number of FCM Buffers configuration parameter - fcm_num_buffers" on page 502
- "Sort Heap Threshold configuration parameter - sheapthres" on page 406
- "Application Support Layer Heap Size configuration parameter - aslheapsz" on page 417
- "Package Cache Size configuration parameter - pckcachesz" on page 399
- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Database Heap configuration parameter - dbheap" on page 392
- "Utility Heap Size configuration parameter - util_heap_sz" on page 396
- "Catalog Cache Size configuration parameter - catalogcache_sz" on page 393
- "Maximum Size of Application Group Memory Set configuration parameter - appgroup_mem_sz" on page 402
- "Percent of Memory for Application Group Heap configuration parameter - groupheap_ratio" on page 403

Buffer pools

Buffer pools are a critically important memory component. This section describes buffer pools and provides information about managing them for good performance.

Buffer-pool management

A buffer pool is memory used to cache table and index data pages as they are being read from disk, or being modified. The buffer pool improves database system performance by allowing data to be accessed from memory instead of from disk. Because memory access is much faster than disk access, the less often the database manager needs to read from or write to a disk, the better the performance. Because most data manipulation takes place in buffer pools, configuring buffer pools is the single most important tuning area. Only large objects and long field data are not manipulated in a buffer pool.

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 database manager looks for it in the buffer pool. If the requested data is in the buffer pool, it can be retrieved without disk access, resulting in faster performance.

Memory is allocated for the buffer pool when a database is activated or when the first application connects to the database. Buffer pools can also be created, dropped, and resized while the database is manager is running. If you use the IMMEDIATE keyword when you use ALTER BUFFERPOOL to increase the size of the buffer pool, memory is allocated as soon as you enter the command if the memory is available. If the memory is not available, the changed occurs when all applications are disconnected and the database is reactivated. If you decrease the size of the buffer pool, memory is deallocated at commit time. When all applications are disconnected, the buffer-pool memory is de-allocated.

Note: To reduce the necessity of increasing the size of the *dbheap* database configuration parameter when buffer-pool sizes increase, nearly all buffer-pool memory, which includes page descriptors, buffer-pool descriptors, and the hash tables, comes out of the database shared memory set and is sized automatically.

To ensure that an appropriate buffer pool is available in all circumstances, DB2® creates small buffer pools, one with each page size: 4K, 8K, 16K, and 32K. The size of each buffer pool is 16 pages. These buffer pools are hidden from the user. They are not present in the system catalogs or in the buffer pool system files. You cannot use or alter them directly, but DB2 uses these buffer pools in the following circumstances:

- When a buffer pool of the required page size is inactive because not enough memory was available to create it after a CREATE BUFFERPOOL statement was executed with the IMMEDIATE keyword.

A message is written to the administration notification log. If necessary, table spaces are remapped to a hidden buffer pool. Performance might be drastically reduced.

- When the ordinary buffer pools cannot be brought up during a database connect

  This problem is likely to have a serious cause, such as out-of-memory condition. Although DB2 will be fully functional because of the hidden buffer pools, performance will degrade drastically. You should address this problem immediately. You receive a warning when this occurs and a message is written to the administration notification log.

Pages remain in the buffer pool until the database is shut down, or until the space occupied by a page is required for another page. The following criteria determine which page is removed to bring in another page:

- How recently the page was referenced
- The probability that the page will be referenced again by the last agent that looked at it
- The type of data on the page
- Whether the page was changed in memory but not written out to disk (Changed pages are always written to disk before being overwritten.)

In order for pages to be accessed from memory again, changed pages are not removed from the buffer pool after they are written out to disk unless the space is needed.

When you create a buffer pool, the default page size is 4 KB but you can specify a page size of 4 KB, 8 KB, 16 KB, or 32 KB. Because pages can be read into a buffer pool only if the table-space page size is the same as the buffer-pool page size, the page size of your table spaces should determine the page size that you specify for buffer pools. You cannot alter the page size of the buffer pool after you create it. You must create a new buffer pool with a different page size.

Note: On 32-bit platforms that run Windows® NT, you can create large buffer pools if you have enabled Address Windowing Extensions (AWE) or Advanced Server and Data Center Server on Windows 2000.

Related concepts:

- “Organization of memory use” on page 251
- “Secondary buffer pools in extended memory on 32-bit platforms” on page 266
- “Buffer-pool management of data pages” on page 267
- “Illustration of buffer-pool data-page management” on page 269

• “Management of multiple database buffer pools” on page 271

Secondary buffer pools in extended memory on 32-bit platforms

On 64-bit platforms, large virtual-addressable memory can be accessed in the normal way, without using special techniques. On 32-bit platforms, however, virtual addressable memory is usually limited to between 2 GB and 4 GB. If your 32-bit machine has more real addressable memory than the maximum amount, you can configure any additional real addressable memory beyond virtual addressable memory as an *extended storage cache*. Any of the defined buffer pools can use an extended storage cache to improve performance. You define the extended storage cache as a number of memory segments.

If you define some of the real addressable memory as an extended storage cache, this memory can no longer be used for other purposes, such as a JFS-cache or as process private address space. More system paging might occur if you allocate real addressable memory to an extended storage cache.

The buffer pools perform the first-level caching, and any extended storage cache is used by the buffer pools as secondary-level caching. Ideally, the buffer pools hold the data that is most frequently accessed, while the extended storage cache hold data that is accessed less frequently.

Note: You can allocate Windows® 2000 Address Windowing Extensions (AWE) buffer pools using the DB2_AWE registry variable. Windows AWE is a set of memory management extensions that allow applications to manipulate memory above certain limits, which depend on the process model of the application. For information, refer to your Windows system documentation. Note, however, that if you use the memory for this purpose you cannot also use the extended storage cache.

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.
- *estore_seg_sz* defines the size of each extended memory segment. This size is determined by the platform on which the extended storage cache is used.

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 use buffer pools defined with different page sizes, any of these buffer pools can be defined to use extended storage. The page size used with extended storage support is the largest of those defined.

Although the database manager cannot directly manipulate data that resides in the extended storage cache, 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 defined associated extended storage cache are allocated when a database is activated or when the first connection occurs.

Related concepts:

- "Buffer-pool management" on page 264
- "Memory management" on page 44

Related reference:

- "Extended Storage Memory Segment Size configuration parameter - estore_seg_sz" on page 437
- "Number of Extended Storage Memory Segments configuration parameter - num_estore_segs" on page 437

Buffer-pool management of data pages

Pages in the buffer pool are either *in use*, *dirty*, or *clean*:

- In-use pages are currently being read or updated. They can be read, but not updated, by other agents.
- "Dirty" pages contain data that has been changed but has not yet been written to disk.
- After a changed page is written to disk, it is clean but remains in the buffer pool until its space is needed for new pages. Clean pages can also be migrated to an associated extended storage cache, if one is defined.

  When the percentage of space occupied by changed pages in the buffer pool exceeds the value specified by the *chngpgs_thresh* configuration parameter, page-cleaner agents begin to write clean buffer pages to disk.

Page-cleaner agents

Page-cleaner agents perform I/O as background processes and allow applications to run faster because their agents can perform actual transaction work. Page-cleaner agents are sometimes referred to as *asynchronous page cleaners* or *asynchronous buffer writers* because they are not coordinated with the work of other agents and work only when required.

To improve performance in update-intensive workloads, configure more page-cleaner agents. Performance improves if more page-cleaner agents are available to write dirty pages to disk. This is also true when there are many data-page or index-page writes in relation to the number of asynchronous data-page or index-page writes.

Page cleaning and fast recovery

If more pages have been written to disk, recovery of the database is faster after a system crash because the database manager can rebuild more of the buffer pool from disk instead of having to replay transactions from the database log files. For this reason, page cleaning is performed 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
- *softmax* represents the percentage of log files to be recovered following a database crash.

  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.

To minimize log read time during recovery, use the database system monitor to track the number of times that page cleaning is performed. The system monitor *pool_lsn_gap_clns (buffer pool log space cleaners triggered)* monitor element provides this information.

The size of the log that must be read during recovery is the difference between the location of the following records in the log:

- The most recently written log record
- The log record that describes the oldest change to data in the buffer pool.

Related concepts:

- "Illustration of buffer-pool data-page management" on page 269

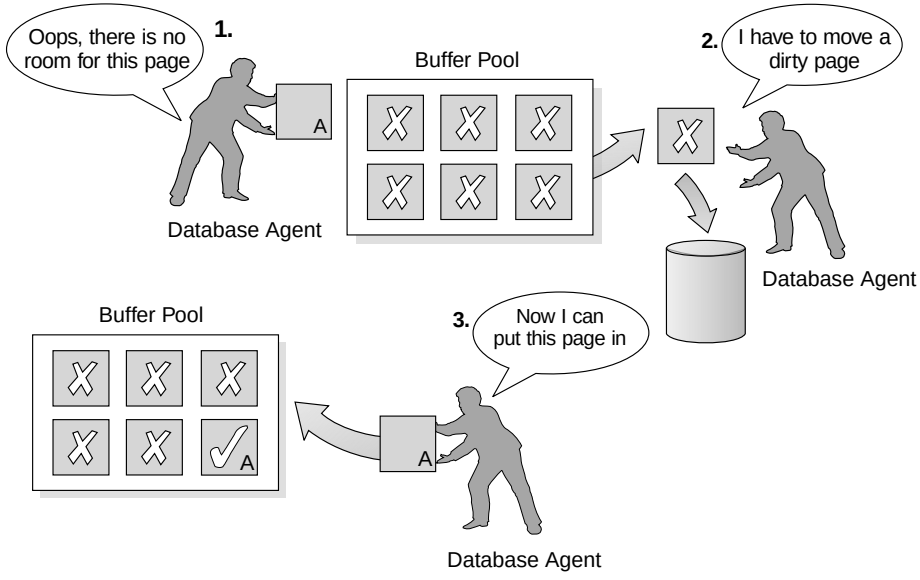
Related reference:

- “Changed Pages Threshold configuration parameter - chngpgs_thresh” on page 431

Illustration of buffer-pool data-page management

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.

Without Page Cleaners



With Page Cleaners

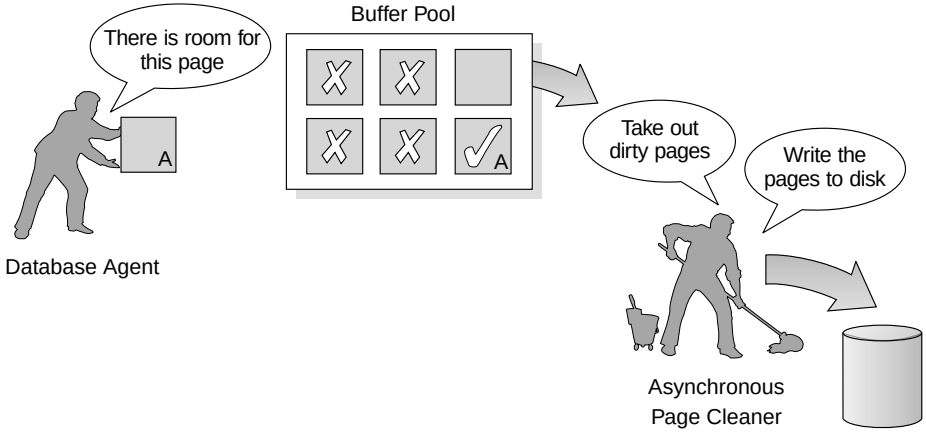


Figure 23. Asynchronous page cleaner. "Dirty" pages are written out to disk.

Related concepts:

- "Buffer-pool management" on page 264
- "Buffer-pool management of data pages" on page 267

Management of multiple database buffer pools

Although each database requires at least one buffer pool, you might create several buffer pools, each of a different size or with a different page size, for a single database that has table spaces of more than one page size. Each buffer pool has a minimum size, which depends on the platform.

A new database has a default buffer pool called IBMDEFAULTBP with a size determined by the platform and a default page size of 4 KB. 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. You can resize the default buffer pool and change its attributes, but you cannot drop it.

Note: During normal database manager operation, you can use the ALTER BUFFERPOOL command to resize a buffer pool.

Page sizes for buffer pools

After you create or migrate a database, you can create other buffer pools. For example, when planning your database, you might have determined that 8 KB page sizes were best for tables. As a result, you should create a buffer pool with an 8 KB page size as well as one or more table spaces with the same page size. You cannot use the ALTER TABLESPACE statement to assign a table space to a buffer pool that uses a different page size.

Note: If you create a table space with a page size greater than 4 KB, such as 8 KB, 16 KB, or 32 KB, you need to assign it to a buffer pool that uses the same page size. If this buffer pool is currently not active, DB2® attempts to assign the table space temporarily to another active buffer pool that uses the same page size if one or to one of the default "hidden" buffer pools that DB2 creates when the first client connects to the database. When the database is activated again, and the originally specified buffer pool is active, then DB2 assigns the table space to that buffer pool.

When you create a buffer pool, you specify the size of the buffer pool as a required parameter of the DDL statement CREATE BUFFERPOOL. To increase or decrease the buffer-pool size later, use the DDL statement ALTER BUFFERPOOL.

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 size was changed by the ALTER BUFFERPOOL statement for a particular database partition.

Advantages of large buffer pools

Large buffer pools provide the following advantages:

- Enable frequently requested data pages to be kept in the buffer pool, which allows 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 that contain the temporary table spaces.

Advantages of 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.

In all other circumstances, consider using more than one buffer pool for the following reasons:

- Temporary table spaces can be assigned to a separate buffer pool to provide better performance for queries that require temporary storage, especially sort-intensive queries.
- If data must be accessed repeatedly and quickly by many short update-transaction applications, consider assigning the table space that contains the data to 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 that frequent updates on the first set of tables have on 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 for an application that requires very random access into a very large table. In such a case, data need not be kept in the

buffer pool for longer than a single query. It is better to keep a small buffer pool for this data, and free the extra memory for other uses, such as 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.

Buffer-pool memory allocation at startup

When you use the CREATE BUFFERPOOL command to create a buffer pool or use the ALTER BUFFERPOOL command to alter 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. If you create or modify buffer pools while the database manager is on-line, additional memory should be available in database global memory. If you specify the IMMEDIATE keyword when you create a new buffer pool or increase the size of an existing buffer pool and the required memory is not available, the database manager makes the change the next time the database is activated. On 32-bit platforms, the memory must be available and can be reserved in the global database memory, as described in detailed information for the *database_memory* database configuration parameter.

If this memory is not available when a database starts, the database manager attempts to start one of each buffer pool defined with a different page size. However, the buffer pools are started only with a minimal size of 16 pages each. To specify a different minimal buffer-pool size, use the DB2_OVERRIDE_BPF registry variable . Whenever a buffer pool cannot be allocated at startup, an SQL1478W (SQLSTATE 01626) warning is returned. The database continues in this operational state until its configuration is changed and the database can be fully restarted.

The database manager starts with minimal-sized values only to allow you to connect to the database so that you can reconfigure the buffer pool sizes or perform other critical tasks. As soon as you perform these tasks, restart the database. Do not operate the database for an extended time in such a state.

Related concepts:

- “Buffer-pool management” on page 264
- “Secondary buffer pools in extended memory on 32-bit platforms” on page 266

Prefetching concepts

Prefetching data into the buffer pools usually improves performance by reducing the number of disk accesses and retaining frequently accessed data in memory.

Prefetching data into the buffer pool

Prefetching pages means that one or more pages are retrieved from disk in the expectation that they will be required by an application. Prefetching index and data pages into the buffer pool can help improve performance by reducing the I/O wait time. In addition, parallel I/O enhances prefetching efficiency.

There are two categories of prefetching:

- **Sequential prefetch**: A mechanism that reads consecutive pages into the buffer pool before the pages are required by the application.
- **List prefetch**: Sometimes called *list sequential prefetch*. Prefetches a set of non-consecutive data pages efficiently.

These two methods of reading data pages are in addition to a normal read. A normal read is used when only one or a few consecutive pages are retrieved. During a normal read, one page of data is transferred.

Prefetching and Intra-Partition Parallelism

Prefetching is important to the performance of intra-partition parallelism, which uses multiple subagents when scanning an index or a table. Such 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 for a serial scan, the query runs more slowly because the agent always needs to wait for I/O. If prefetching does not occur for a parallel scan, all subagents might need to wait because one subagent 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.

Related concepts:

- "Buffer-pool management" on page 264
- "Sequential prefetching" on page 275

- "List prefetching" on page 278
- "I/O server configuration for prefetching and parallelism" on page 279
- "Illustration of prefetching with parallel I/O" on page 280

Sequential prefetching

Reading several consecutive pages into the buffer pool using a single I/O operation can greatly reduce your application overhead. In addition, multiple parallel I/O operations to read several ranges of pages into the buffer pool can help reduce I/O wait time.

Prefetching starts when the database manager determines that sequential I/O is appropriate and that prefetching might 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, which probably requires a table scan, would be a good candidate for sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
```

Implications of the PREFETCHSIZE for table spaces

To define the number of prefetched pages for each table space, use the PREFETCHSIZE clause in either the CREATE TABLESPACE or ALTER TABLESPACE statements. The value that you specify 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 number of table space containers and the EXTENTSIZE value for your table space, which is the number of pages that the database manager writes to a container before it uses a different container. For example, if the extent size is 16 pages and the table space has two containers, you might set the prefetch quantity to 32 pages.

The database manager monitors buffer-pool usage to ensure that prefetching does not remove pages from the buffer pool if another unit of work needs them. To avoid problems, the database manager can limit the number of prefetched pages to less than you specify for the table space.

The prefetch size can have significant performance implications, particularly for large table scans. Use the database system monitor and other system monitor tools to help you tune PREFETCHSIZE for your table spaces. You might 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.

If there are I/O waits and the query is prefetching data, you might increase the value of PREFETCHSIZE. If the prefetcher is not the cause of the I/O wait, increasing the PREFETCHSIZE value will not improve the performance of your query.

In all types of prefetch, multiple I/O operations might 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, configure the containers to use separate physical devices.

Sequential detection

In some cases it is not immediately obvious that sequential prefetch will improve performance. In these cases, the database manager can monitor I/O and activate prefetching if sequential page reading is occurring. In this case, prefetching is activated and deactivated by the database manager as appropriate. This type of sequential prefetch is known as *sequential detection* and applies to both index and data pages. Use the *seqdetect* configuration parameter to control whether the database manager performs sequential detection.

For example, if sequential detection is turned on, the following SQL statement might benefit from sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
WHERE EMPNO BETWEEN 100 AND 3000
```

In this example, the optimizer might have started 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 might improve performance, so data-page prefetch will occur.

Index-page prefetch might also occur in this example. If many index pages must be examined and the database manager detects that sequential page reading of the index pages is occurring, then index-page prefetching occurs.

Related concepts:

- “Buffer-pool management” on page 264
- “Prefetching data into the buffer pool” on page 274
- “List prefetching” on page 278
- “Block-based buffer pools for improved sequential prefetching” on page 277

Block-based buffer pools for improved sequential prefetching

Prefetching pages from disk is expensive because of I/O overhead. Throughput can be significantly improved if processing is overlapped with I/O. Most platforms provide high-performance primitives that read contiguous pages from disk into non-contiguous portions of memory. These primitives are usually called *scattered read* or *vectored I/O*. On some platforms, performance of these primitives cannot compete with doing I/O in large block sizes.

By default, the buffer pools are page-based, which means that contiguous pages on disk are prefetched into non-contiguous pages in memory. Sequential prefetching can be enhanced if contiguous pages can be read from disk into contiguous pages within a buffer pool.

You can create block-based buffer pools for this purpose. A block-based buffer pool consist of both a page area and a block area. The page area is required for non-sequential prefetching workloads. The block area consist of blocks where each block contains a specified number of contiguous pages, which is referred to as the *block size*.

The optimal usage of a block-based buffer pool depends on the specified block size. The block size is the granularity at which I/O servers doing sequential prefetching consider doing block-based I/O. The extent is the granularity at which table spaces are striped across containers. Because multiple table spaces with different extent sizes can be bound to a buffer pool defined with the same block size, consider how the extent size and the block size interact for efficient use of buffer-pool memory. Buffer-pool memory can be wasted in the following circumstances:

- If the extent size, which determines the prefetch request size, is smaller than BLOCK_SIZE specified for the buffer pool.
- If some pages requested in the prefetch request are already present in the page area of the buffer pool.

The I/O server allows some wasted pages in each buffer-pool block, but if too much of a block would be wasted, the I/O server does non-block-based prefetching into the page area of the buffer pool. This is not optimal performance.

For optimal performance, bind table spaces of the same extent size to a buffer pool with a block size that equals the table-space extent size. Good performance can be achieved if the extent size is larger than the block size, but not when the extent size is smaller than the block size.

To create block-based buffer pools, use the CREATE and ALTER BUFFERPOOL statements. Block-based buffer pools have the following limitations:

- A buffer pool cannot be made block-based and use extended storage simultaneously.
- Block-based I/O and AWE support cannot be used by a buffer pool simultaneously. AWE support takes precedence over block-based I/O support when both are enabled for a given buffer pool. In this situation, the block-based I/O support is disabled for the buffer pool. It is re-enabled when the AWE support is disabled.

Note: Block-based buffer pools are intended for sequential prefetching. If your applications do not use sequential prefetching, then the block area of the buffer pool is wasted.

Related concepts:

- "Buffer-pool management" on page 264
- "Prefetching data into the buffer pool" on page 274
- "Sequential prefetching" on page 275

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.

If the optimizer uses an index to access rows, it can defer reading the data pages until all the row identifiers (RIDs) have been obtained from the index. For example, the optimizer could perform an index scan to determine the rows and data pages to retrieve, 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'
```

If the data is not clustered according to this index, list prefetch includes a step that sorts the list of RIDs obtained from the index scan.

Related concepts:

- "Buffer-pool management" on page 264
- "Prefetching data into the buffer pool" on page 274

• "Sequential prefetching" on page 275

I/O management

This section describes how to tune I/O servers for the best performance.

I/O server configuration for prefetching and parallelism

To enable prefetching, the database manager starts separate threads of control, known as *I/O servers*, to read data pages. 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 query. The possible prefetch methods determine when and how the database manager generates the prefetch requests.

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. To maximize the opportunity for parallel I/O, set *num_ioservers* to at least the number of physical disks in the database.

It is better to overestimate the number of I/O servers than to underestimate. If you specify extra I/O servers, these servers are not used, and their memory pages are paged out. As a result, performance does not suffer. Each I/O server process is numbered. The database manager always uses the lowest numbered process, so some of the upper numbered processes might never be used.

To estimate the number of I/O servers that you might need, consider the following:

- The number of database agents that could be writing prefetch requests to the I/O server queue concurrently.
- The highest degree to which the I/O servers can work in parallel.

Configuration for asynchronous I/O

On some platforms, DB2® uses asynchronous I/O (AIO) to improve performance of activities such as page cleaning and prefetching. AIO is most effective if data in containers is distributed across multiple disks. Performance also benefits from tuning the underlying operating system AIO infrastructure.

For example, on AIX, you might tune AIO on the operating system. When AIO works on either SMS or DMS file containers, operating system processes called AIO servers manage the I/O. A small number of such servers might

restrict the benefit of AIO by limiting the number of AIO requests. To configure the number of AIO servers on AIX, use the *smit* AIO *minservers* and *maxservers* parameters.

Related concepts:

- "Parallel processing for applications" on page 107
- "Illustration of prefetching with parallel I/O" on page 280
- "Parallel I/O management" on page 282

Related reference:

- "Number of I/O Servers configuration parameter - num_ioservers" on page 434

Illustration of prefetching with parallel I/O

The following figure illustrates how I/O servers are used to prefetch data into a buffer pool.

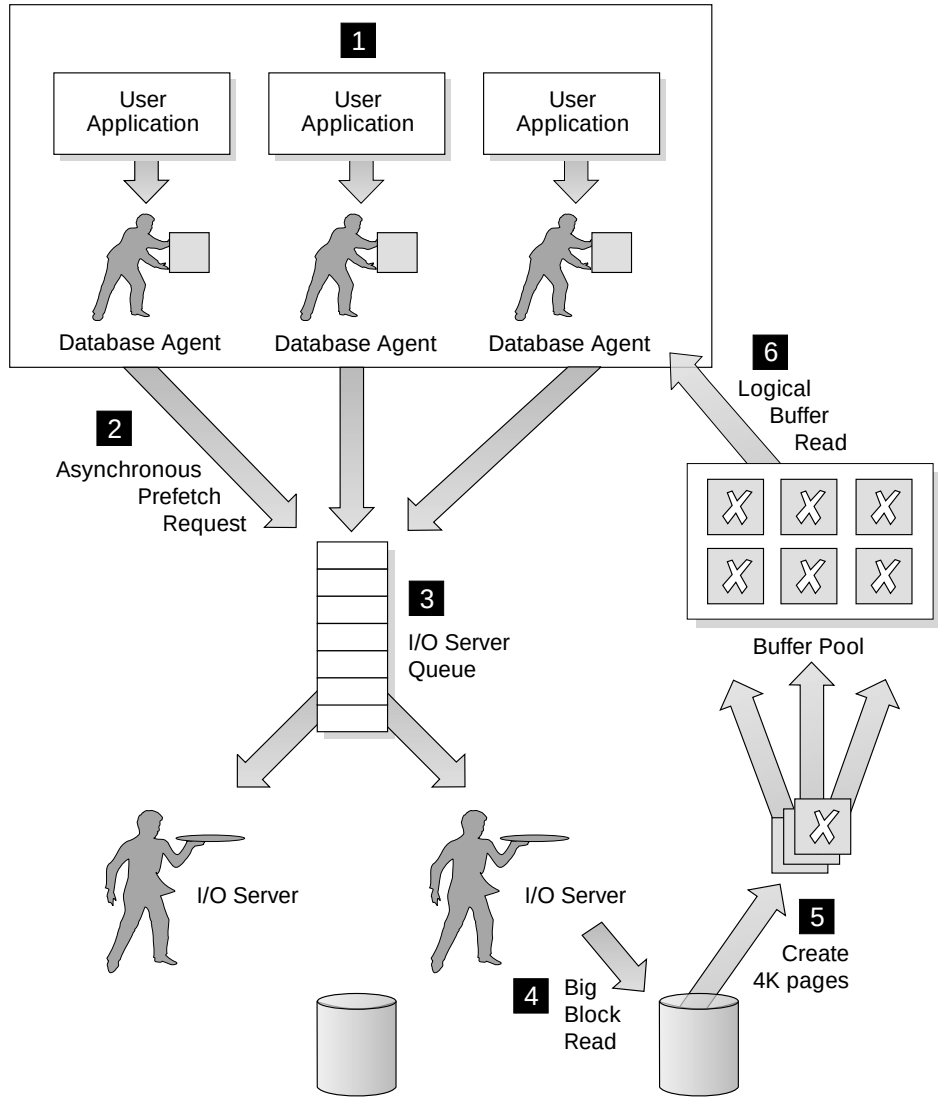


Figure 24. Prefetching data using I/O servers

1 The user application passes the SQL request to the database agent that has been assigned to the user application by the database manager.

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 reads the prefetch request from the

queue and then reads the data from the table space into the buffer pool. The number of I/O servers that can fetch data from a table space at the same time depends on the number of prefetch requests in the queue and the number of I/O servers configured by the *num_ioservers* database configuration parameter.

6 The database agent performs the necessary operations on the data pages in the buffer pool and returns the result to the user application.

Related concepts:

- "Prefetching data into the buffer pool" on page 274
- "Sequential prefetching" on page 275
- "List prefetching" on page 278
- "I/O server configuration for prefetching and parallelism" on page 279
- "Parallel I/O management" on page 282
- "Agents in a partitioned database" on page 315

Parallel I/O management

If multiple containers exist for a table space, the database manager can initiate *parallel I/O*, in which database manager uses multiple I/O servers to process the I/O requirements of a single query. Each I/O server processes the I/O workload for a separate container, so that several containers can be read in parallel. Performing I/O in parallel can result in significant improvements to I/O throughput.

Although a separate I/O server can handle the workload for each container, the actual number of I/O servers that can perform I/O in parallel is limited to the number of physical devices over which the requested data is spread. For this reason, you need as many I/O servers as physical devices.

Parallel I/O is initiated differently in the following cases:

- **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 many small requests along the extent boundaries. These small 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 the same way 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 you load data, you can specify the level of I/O parallelism with the LOAD command DISK_PARALLELISM option. If you do not specify this option, the database manager uses a default value 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. Specify slightly more I/O servers 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. To prevent over-use of the buffer pool, prefetch size should not be too large. 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 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 reduce the potential for optimized parallel prefetch. Consider the following examples:

  – After a smaller container is filled, 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 might 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 is the only container to store additional data. The database manager cannot use parallel prefetch to access this additional data.

- There is adequate I/O capacity when using intra-partition parallelism. On SMP machines, intra-partition parallelism can reduce the elapsed time for query by running the query on multiple processors. Sufficient I/O capacity is required to keep each processor busy. Usually additional physical drives are required to provide the I/O capacity.

The prefetch size must be larger for prefetching to occur at higher rates and use I/O capacity effectively.

The number of physical drives required depends on the speed and capacity of the drives and the I/O bus and on the speed of the processors.

Related concepts:

- "I/O server configuration for prefetching and parallelism" on page 279
- "Illustration of prefetching with parallel I/O" on page 280
- "Guidelines for sort performance" on page 284

Guidelines for sort performance

Because queries often require sorted or grouped results, sorting is often required, and the proper configuration of the sort heap areas is crucial to good query 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
- An index is created.
- An index is dropped, which causes index page numbers to be sorted.

Sorting involves two steps:

1. A sort phase

   A sort can be *overflowed* or *non-overflowed*. If the sorted data cannot fit entirely into the sort heap, which is 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.

2. Return of the results of the sort phase.

   The return can be *piped* or *non-piped*. If sorted information can return directly without requiring a temporary table to store a final, sorted list of data, it is a piped sort. If the sorted information requires a temporary table to be returned, it is a non-piped sort . A piped sort always performs better than a non-piped sort.

Elements that affect sorting

The following elements affect sort performance:

- The settings for the following database configuration parameters:
  - Sort heap size, (*sortheap*), which specifies the amount of memory to be used for each sort
  - Sort heap threshold (*sheapthres*) and the sort heap threshold for shared sorts (*sheapthres_shr*), which control 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.

To find out if you have a sort performance problem, look at the total CPU time spent sorting compared to the time spent for the whole application. You can use the database system monitor to see this information, but the Performance Monitor shows *total sort time* by default, as well as other times such as *I/O* and *lock wait*. The Performance Monitor is made up of the "Snapshot Monitor" and "Event Monitor" and is available from the Control Center.

If total sort time is a large portion CPU time for the application, 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 find out 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, overall sort memory available across the instance (*sheapthres*) should be as large as possible without causing excessive paging. Although a sort can be performed entirely in sort memory, this might cause excessive page swapping. In this case, you lose the advantage of a large sort heap. For this reason, you should use an operating system monitor to track changes in system paging whenever you adjust the sorting configuration parameters.

Also note that in a piped sort, the sort heap is not freed until the application closes the cursor associated with that sort. A piped sort can continue to use up memory until the cursor is closed.

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 long keys are used for sorts, increase the *sortheap* configuration parameter.

Techniques for managing sorting performance

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 the queries that have sort operations.

You can use the database system monitor and benchmarking techniques to help set the *sortheap* and *sheapthres* configuration parameters. 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

  The performance details view of the Snapshot Monitor shows these performance variables.
- Set *sortheap* to the average *total sort heap in use* for each database.
- Set the *sheapthres*. To estimate an appropriate size:
  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.

Related reference:

- "Sort Heap Size configuration parameter - sortheap" on page 405
- "Sort Heap Threshold configuration parameter - sheapthres" on page 406
- "Sort Heap Threshold for Shared Sorts configuration parameter - sheapthres_shr" on page 408

Table management

This section describes methods of managing tables for performance improvements.

Table reorganization

After many changes to table data, logically sequential data may be on non-sequential physical data pages so that the database manager must perform additional read operations to access data. Additional read operations are also required if a significant number of rows have been deleted. In such a case, you might consider reorganizing the table to match the index and to reclaim space. You can reorganize the system catalog tables as well as database tables.

Note: Because reorganizing a table usually takes more time than running statistics, you might execute RUNSTATS to refresh the current statistics for your data and rebind your applications. If refreshed statistics do not improve performance, reorganization might help. For detailed information about the options and behavior of the REORG TABLE utility, refer to its command reference.

Consider the following factors, which might indicate that you should reorganize a table:

- A high volume of insert, update, and delete activity on tables accessed by queries
- Significant changes in the performance of queries that use an index with a high cluster ratio
- Executing RUNSTATS to refresh statistical information does not improve performance
- The REORGCHK command indicates a need to reorganize your table
- The tradeoff between the cost of increasing degradation of query performance and the cost of reorganizing your table, which includes the CPU time, the elapsed time, and the reduced concurrency resulting from the REORG utility locking the table until the reorganization is complete.

Reducing the need to reorganize tables

To reduce the need for reorganizing a table, perform these tasks after you create the table:

- Alter table to add PCTFREE
- Create clustering index with PCTFREE on index
- Sort the data
- Load the data

After you have performed these tasks, the table with its clustering index and the setting of PCTFREE on table helps preserve the original sorted order. If enough space is allowed in table pages, new data can be inserted on the correct pages to maintain the clustering characteristics of the index. As more data is inserted and the pages of the table become full, records are appended to the end of the table so that the table gradually becomes unclustered.

If you perform a REORG TABLE or a sort and LOAD after you create a clustering index, the index attempts to maintain a particular order of data, which improves the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility.

Note: Creating multi-dimensional clustering (MDC) tables might reduce the need to reorganize tables. For MDC tables, clustering is maintained on the columns that you specify as arguments to the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. However, REORGCHK might recommend reorganization of an MDC table if it considers that there are too many unused blocks or that blocks should be compacted.

Related concepts:

- "Index reorganization" on page 303
- "DMS device considerations" on page 307
- "SMS table spaces" on page 19
- "DMS table spaces" on page 20
- "Table and index management for standard tables" on page 23
- "Snapshot monitor" in the *System Monitor Guide and Reference*
- "Table and index management for MDC tables" on page 28

Related tasks:

- "Determining when to reorganize tables" on page 288
- "Choosing a table reorganization method" on page 291

Determining when to reorganize tables

To determine how performance is related to changes in database tables and indexes, examine the statistics collected by RUNSTATS. Among other information, the statistics show the data distribution within a table, the number of used and empty pages, and RIDs marked deleted in index leaf pages. Statistics also provide information about prefetch efficiency. If you run RUNSTATS regularly and analyze the statistics over a period of time, you can identify performance trends.

In particular, analyzing the statistics produced by RUNSTATS can indicate when and what kind of reorganization is necessary.

Note: The REORGCHK command also returns information about some of the catalog statistics data and can advise you about whether tables need to be reorganized. However, running specific queries against the catalog statistics tables at regular intervals or specific times can provide a performance history that allows you to spot trends that might have wider implications for performance.

Procedure:

To determine whether you need to reorganize tables, query the catalog statistics tables and monitor the following statistics:

1. Overflow of rows

   Query the OVERFLOW column in the SYSSTAT.TABLES table to monitor the overflow number. This column stores the number of rows that do not fit on their original pages. Row data can overflow when VARCHAR columns are updated with longer values. In such cases, a pointer is kept at the original location in the row original location. This can hurt degrades performance because the database manager must follow the pointer to find the contents of the row. This two-step process increases the processing time and might also increase the number of I/Os.

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

2. Fetch statistics

   Query the three following columns in the SYSCAT.INDEXES and SYSSTAT.INDEXES catalog statistics tables to determine the effectiveness of the prefetchers when the table is accessed in index order. These statistics characterize the the average performance of the prefetchers against the underlying table.

   - The AVERAGE_SEQUENCE_FETCH_PAGES column stores the average number of adjoining pages in sequence in the table. These pages would be eligible for prefetching. A small number indicates that the prefetchers cannot be effective because they never achieve the full prefetching size configured for the tablespace. A large number indicates that the prefetchers are performing effectively. This number should approach NPAGES for a clustered index and table.
   - The AVERAGE_RANDOM_FETCH_PAGES column stores the average number of pages that must be accessed randomly while scanning pages that are primarily in sequence. The prefetchers ignore small numbers of random pages when most pages are in sequence, and continue to prefetch to the configured prefetch size. As the number of random fetch

pages increases, the table is becoming more disorganized. Such disorganization is usually caused by inserts that occur out of sequence, either at the end of the table or in overflow pages. This causes fetches that slow query performance when the index is used to access a range of values.

- The AVERAGE_SEQUENCE_FETCH_GAP column stores the average number of pages that interrupted prefetching. These occur when many pages are accessed randomly, which interrupts the prefetchers. A large number indicates a table that is disorganized or poorly clustered to the index. A small number indicates a clustered index and table.

3. Number of index leaf pages that contain RIDs marked deleted but not removed

   In type-2 indexes, RIDs are not usually physically deleted when the RID is marked deleted. In such indexes, useful space might be occupied by these logically deleted RIDs. Query the NUM_EMPTY_LEAFS column of the SYSCAT.INDEXES and SYSSTAT.INDEXES statistics tables to retrieve the number of leaf pages that contain on which all RIDS are logically deleted. For leaf pages on which not all RIDs are marked deleted, the total number of logically deleted RIDs is stored in the NUMRIDS_DELETED column. Use this information to estimate how much space might be reclaimed by executing REORG INDEXES with the CLEANUP ALL option. To reclaim only the space in pages on which all RIDs are marked deleted, execute REORG INDEXES with the CLEANUP ONLY PAGES options.

4. Custer-ratio and cluster-factor statistics for indexes

   A cluster-ratio statistic between 0 to 100 is stored in the CLUSTERRATIO column of the SYSTCAT.INDEXES catalog table. If you collect DETAILED index statistics, a finer cluster-factor statistic between 0 and 1 is stored in the CLUSTERFACTOR column. Only one of these two clustering statistics can be recorded in the SYSCAT.INDEXES catalog table. In general, only one of the indexes in a table can have a high degree of clustering. A value of -1 indicates that no statistics for clustering are available. To compare the cluster-factor with the cluster-ratio values, multiply the cluster factor by 100 to obtain a percentage.

   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 might also improve the performance of an unclustered index.

   If table data was initially clustered in the order of a certain index, and the clustering statistics information indicates that the data is now poorly clustered for that same index, reorganize the table to cluster the data again.

5. Number of leaf pages

Query the NLEAF column in the SYSCAT.INDEXES table to monitor 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 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.

Note: By default, ten percent free space is left on each index page when the indexes are rebuilt. To increase the free space amount, specify the PCTFREE parameter when you create the index. Whenever you reorganize the index, the PCTFREE value is used. Free space greater than ten percent might reduce frequency of index reorganization because the additional space can accommodate additional index inserts.

6. Comparison of file pages

   To calculate the number of empty pages in a table, query the FPAGES and NPAGES columns in SYSCAT.TABLES and subtract the NPAGES number from the FPAGES number. The FPAGES column stores the number of pages in use; the NPAGES column stores the number of pages that contain rows. Empty pages can occur when entire ranges of rows are deleted.

   As the number of empty pages increases, the need for a table reorganization also increases. Reorganizing the table reclaims the empty pages to compress the amount of space used by a table. In addition, because empty pages are read into the buffer pool for a table scan, reclaiming unused pages can also improve the performance of a table scan.

Related concepts:

- "Catalog statistics tables" on page 124
- "Table reorganization" on page 287
- "Index reorganization" on page 303

Related tasks:

- "Collecting catalog statistics" on page 120

Choosing a table reorganization method

DB2® provides two methods of reorganizing tables: classic and in-place. In general, classic table reorganization is faster, but should be used only if your applications function without write access to tables during the reorganization. If your environment does not allow this restriction, although in-place reorganization is slower, it can occur in the background while normal data access continues. Consider the features of each method and decide which method is more appropriate for your environment.

Procedure:

To choose a table reorganization method, consider the features of the following methods:

- **Classic table reorganization**

  This method provides the fastest table reorganization, especially if you do not need to reorganize LOB or LONG data. In addition, indexes are rebuilt in perfect order after the table is reorganized. Read-only applications can access the original copy of the table except during the last phases or the reorganization, in which the permanent table replaces the shadow copy of the table and the indexes are rebuilt.

  On the other hand, consider the following possible disadvantages:

  – Large space requirement

    Because classic table reorganization creates the shadow copy of the table, it can require twice as much space as the original table. If the reorganized table is larger than the original, reorganization can require more than twice as much space as the original.

    The shadow copy can be built in a temporary tablespace if the table tablespace is not large enough, but the replace phase performs best in the same DMS table space. Tables in SMS table spaces must always store the shadow copy in temporary space.

  – Limited table access

    Even read-only access is limited to the first phases of the reorganization process.

  – All or nothing process

    If the reorganization fails at any point, it must be restarted from the beginning on the nodes where it failed.

  – Performed within the controller of the application that invokes it

    The reorganization can be stopped only by that application or by a user who understands how to stop the process and has authority to execute the FORCE command for the application.

  Recommendation: Choose this method if you can reorganize tables during a maintenance window.

- **In-place table reorganization**

  The in-place method is slower and does not ensure perfectly ordered data, but it can allow applications to access the table during the reorganization. In addition, in-place table reorganization can be paused and resumed later by anyone with the appropriate authority by using the schema and table name.

Note: In-place table reorganization is allowed only on tables with type-2 indexes and without extended indexes.

Consider the following trade-offs:

– Imperfect index reorganization

  You might need to reorganize indexes later to reduce index fragmentation and reclaim index object space.

– Longer time to complete

  When required, in-place reorganization defers to concurrent applications. This means that long-running statements or RR and RS readers in long-running applications can slow the reorganization progress. In-place reorganization might be faster in an OLTP environment in which many small transactions occur.

– Requires more log space

  Because in-place table reorganization logs its activities so that recovery is possible after an unexpected failure, it requires more log space than classic reorganization.

  It is possible that in-place reorganization will require log space equal to several times the size of the reorganized table. The amount of required space depends on the number of rows that are moved and the number and size of the indexes on the table.

Recommendation: Choose in-place table reorganization for 24x7 operations with minimal maintenance windows.

Refer to the REORG TABLE syntax descriptions for detailed information about executing these table reorganization methods.

Monitoring the progress of table reorganization

Information about the current progress of table reorganization is written to the history file for database activity. The history file contains a record for each reorganization event. To view this file, execute the `db2 list history` command for the database that contains the table you are reorganizing.

You can also use table snapshots to monitor the progress of table reorganization. Table reorganization monitoring data is recorded regardless of the Database Monitor Table Switch setting.

If an error occurs, an SQLCA dump is written to the history file. For an in-place table reorganization, the status is recorded as PAUSED.

Related concepts:

• “Table reorganization” on page 287

- "Index reorganization" on page 303

Related tasks:

- "Determining when to reorganize tables" on page 288

Index management

The following sections describe index reorganization for performance improvements.

Advantages and disadvantages of indexes

Although the optimizer decides whether to use an index to access table data, except in the following case, you must decide which indexes might improve performance and create these indexes. Exceptions are the dimension block indexes and the composite block index that are created automatically for each dimension that you specify when you create a multi-dimensional clustering (MDC) table.

You must also execute the RUNSTATS utility to collect new statistics about the indexes in the following circumstances:

- After you create an index
- After you change the prefetch size

You should also execute the RUNSTATS utility at regular intervals to keep the statistics current. Without up-to-date statistics about indexes, the optimizer cannot determine the best data-access plan for queries.

Note: To determine whether an index is used in a specific package, use the SQL Explain facility. To plan indexes, use the `db2advis` tool to get advice about indexes that might be used by one or more SQL statements.

Advantages of an index over no index

If no index exists on a table, a table scan must be performed for each table referenced in a database query. The larger the table, the longer a table scan takes because a table scan requires each table row to be accessed sequentially. Although a table scan might be more efficient for a complex query that requires most of the rows in a table, for a query that returns only some table rows an index scan can access table rows more efficiently.

The optimizer chooses an index scan if the index columns are referenced in the SELECT statement and 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 that are applied to the index reduce the number of rows to be read from the data pages.

Each index entry contains a search-key value and a pointer to the row containing that value. If you specify the ALLOW REVERSE SCANS parameter in the CREATE INDEX statement, the values can be searched in both ascending and descending order. 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.

In addition to the search-key value and row pointer, an index can contain include columns, which are non-indexed columns in the indexed row. Such columns might make it possible for the optimizer to get required information only from the index, without accessing the table itself.

Note: The existence of an index on the table being queried does not guarantee an ordered result set. Only an ORDER BY clause ensures the order of a result set.

Although indexes can reduce access time significantly, they can also have adverse effects on performance. Before you create indexes, consider the effects of multiple indexes on disk space and processing time:

- Each index requires storage or disk space. The exact amount depends on the size of the table and the size and number of columns 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 the value of 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 for the optimizer to consider, which increases the compilation time.

Choose indexes carefully to address the needs of the application program.

Related concepts:

- “Space requirements for indexes” in the *Administration Guide: Planning*
- “Index planning tips” on page 296
- “Index performance tips” on page 299
- “The Design Advisor” on page 242
- “Table reorganization” on page 287
- “Index reorganization” on page 303

- "Table and index management for standard tables" on page 23
- "Table and index management for MDC tables" on page 28
- "Index cleanup and maintenance" on page 302

Related tasks:

- "Creating an index" in the *Administration Guide: Implementation*
- "Collecting catalog statistics" on page 120
- "Collecting index statistics" on page 123

Index planning tips

The indexes that you create should depend on the data and the queries that access it.

Tip: Use the Design Advisor, which is available from the Control Center, or the `db2advis` tool to find the best indexes for a specific query or for the set of queries that defines a workload.

The following guidelines can help you determine how to create useful indexes for various purposes:

- To avoid some sorts, define primary keys and unique keys, wherever possible, by using the CREATE UNIQUE INDEX statement.
- To improve data-retrieval, add INCLUDE columns to unique indexes. Good candidates are columns that:
  - 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.
- To access small tables efficiently, use indexes to optimize frequent queries to tables with more than a few data pages, as recorded in the NPAGES column in the SYSCAT.TABLES catalog view. You should:
  - 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.
- To search efficiently, decide between ascending and descending ordering of keys depending on the order that will be used most often. Although the values can be searched in reverse direction if you specify the ALLOW REVERSE SCANS parameter in the CREATE INDEX statement, scans in the specified index order perform slightly better than reverse scans.
- To save index maintenance costs and space:
  - 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.

– Do not create indexes arbitrarily on all columns. Unnecessary indexes not only use space, but also cause large prepare times. This is especially important for complex queries, when an optimization class with dynamic programming join enumeration is used.

  Use the following general rule for the typical number of indexes that you define for a table. This number is based on the primary use of your database:
  - For online transaction processing (OLTP) environments, create one or two indexes
  - For read-only query environments, you might create more than five indexes
  - For mixed query and OLTP environments, you might create between two and five indexes.

- To improve performance of delete and update operations on the parent table, create indexes on foreign keys.
- For fast sort operations, create indexes on columns that are frequently used to sort the data.
- To improve join performance with a multiple-column index, if you have more than one choice for the first key column, use the column most often specified with the "=" (equijoin) predicate or the column with the greatest number of distinct values as the first key.
- To help keep newly inserted rows clustered according to an index and avoid page splits, define a clustering index. A clustering index should significantly reduce the need for reorganizing the table.

  Use the PCTFREE keyword when you define the table to specify how much free space should be left on the page to allow inserts to be placed appropriately on pages. You can also specify the `pagefreespace` MODIFIED BY clause of the LOAD command.
- To enable online index defragmentation, use the MINPCTUSED option when you create indexes. MINPCTUSED specifies the threshold for the minimum amount of used space on an index leaf page as well as enabling online index defragmentation. This might reduce the need for reorganization at the cost of a performance penalty during key deletions if these deletions physically remove keys from the index page.

Consider creating an index in the following circumstances:

- Create an index 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 the `WORKDEPT` column contains many duplicate values.

- Create an index on a column or columns to order the rows in the sequence required by the query. 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
  ```

- Create an index with a compound key that names each column referenced in a statement. When an index is specified in this way, data can be retrieved from the index only, which is more efficient than accessing the table.

  For example, consider the following SQL statement:

  ```
  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.

- Create an index with INCLUDE columns 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.

Related concepts:

- “Advantages and disadvantages of indexes” on page 294
- “Index performance tips” on page 299
- “The Design Advisor” on page 242
- “Index reorganization” on page 303
- “Online index defragmentation” on page 305
- “Considerations when creating MDC tables” in the *Administration Guide: Planning*

Index performance tips

Consider the following suggestions for using and managing indexes:

- **Specify parallelism at index creation**

  When you create indexes on large tables hosted by 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.

- **Specify separate table spaces for indexes**

  Indexes can be stored in a different table space from the table data. This can allow for more efficient use of disk storage by reducing the movement of read/write heads during index access. You can also create index table spaces on faster physical devices. In addition, you can assign the index table space to a different buffer pool, which might keep the index pages in the buffer longer because they do not compete with table data pages.

  When you do not place indexes 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 can select different values for all the characteristics of a table space. Because indexes are usually smaller than tables and are spread over fewer containers, indexes often have smaller extent sizes, such as 8 and 16 pages. The SQL optimizer considers the speed of the device for a table space when it chooses an access plan.

- **Ensure the degree of clustering**

  If your SQL statement requires ordering, such as ORDER BY, GROUP BY, and DISTINCT, even though an index might satisfy the ordering, the optimizer might not choose the index in the following cases:
  - Index clustering is poor. For information, examine the CLUSTERRATIO and CLUSTERFACTOR columns of SYSCAT.INDEXES.
  - The table is so small that it is cheaper to scan the table and sort the answer set in memory.
  - There are competing indexes for accessing the table.

  After you create a clustering index, perform a REORG TABLE in classic mode, which creates a perfectly organized index. To recluster the table, you might perform a sort and LOAD instead. In general, a table can only be clustered on one index. Build additional indexes after you build the clustering index. A clustering index attempts to maintain a particular order of data, improving the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility.

  To help maintain the clustering ratio, specify an appropriate PCTFREE when you alter a table before you load or reorganize that table. The free space on each page specified by PCTFREE provides space for inserts, so

that these inserts can be clustered appropriately. If you do not specify PCTFREE for the table, reorganization eliminates all extra space.

Note: Clustering is not currently maintained during updates. That is, if you update a record so that its key value changes in the clustering index, the record is not necessarily moved to a new page to maintain the clustering order. To maintain clustering, use DELETE and then INSERT instead of UPDATE.

- **Keep table and index statistics up-to-date**

  After you create a new index, run the RUNSTATS utility to collect index statistics. These statistics allow the optimizer to determine whether using the index can improve access performance.

- **Enable online index defragmentation**

  Online index defragmentation is enabled if the MINPCTUSED clause is set to greater than zero for the index. Online index defragmentation allows indexes to be compacted by merging leaf pages when the free space on a page falls at or below the specified level while the index remains available.

- **Reorganize indexes as necessary**

  To get the best performance from your indexes, consider reorganizing your indexes periodically because updates to tables can cause index page prefetch to become less effective.

  To reorganize the index, either drop it and re-create it or use the REORG utility.

  To reduce the need for frequent reorganization, when you create an index specify an appropriate PCTFREE to leave a percentage of free space on each index leaf page as it is created. During future activity, records can be inserted into the index with less likelihood of causing index page splits. Page splits cause index pages not to be contiguous or sequential, which in turn results in decreased efficiency of index page prefetching.

  Note: The PCTFREE specified when you create the index is retained when the index is reorganized.

  Dropping and re-creating or reorganizing the index also creates a new set of pages that are roughly contiguous and sequential and improves index page prefetch. Although more costly in time and resources, the REORG TABLE utility also ensures clustering of the data pages. Clustering has greater benefit for index scans that access a significant number of data pages.

  In a symmetric multi-processor (SMP) environment, if the *intra_parallel* database manager configuration parameter is YES or ANY, the “classic” REORG TABLE mode, which uses a shadow table for fast table reorganization, can use multiple processors to rebuild the indexes.

- **Analyze EXPLAIN information about index usage**

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

EXPLAIN information also lets you see if table scans on large tables are processed as the inner table of nested loop joins. If they are, an index on the join-predicate column is either missing or considered ineffective for applying the join predicate.

- **Use volatile tables for tables that vary widely in size**

  A *volatile* table is a table that might vary in size at run time from empty to very large. For this kind of table, in which the cardinality varies greatly, the optimizer might generate an access plan that favors a table scan instead of an index scan.

  Declaring a table "volatile" using the ALTER TABLE...VOLATILE statement allows the optimizer to use an index scan on the volatile table. The optimizer will use an index scan instead of a table scan regardless of the statistics in the following circumstances:

  – All columns referenced are in the index
  – The index can apply a predicate in the index scan.

  If the table is a typed table, using the ALTER TABLE...VOLATILE statement is supported only on the root table of the typed table hierarchy.

Related concepts:

- "Advantages and disadvantages of indexes" on page 294
- "Index planning tips" on page 296
- "Index structure" on page 31
- "Index access and cluster ratios" on page 183
- "Table reorganization" on page 287
- "Index reorganization" on page 303
- "Table and index management for standard tables" on page 23
- "Online index defragmentation" on page 305
- "Table and index management for MDC tables" on page 28
- "Index cleanup and maintenance" on page 302

Related reference:

- "Enable Intra-Partition Parallelism configuration parameter - intra_parallel" on page 506

Index cleanup and maintenance

After you create indexes, performance degrades unless you keep the index compact and organized. Consider the following suggestions to keep indexes as small and efficient as possible:

- Enable online index defragmentation

  Create indexes with the MINPCTUSED clause. Drop and recreate existing indexes, if necessary.
- Perform frequent COMMITs or get X locks on tables, either explicitly or by lock escalation, if frequent COMMITS are not possible.

  Index keys marked deleted can be physically removed from the table after the COMMIT. X locks on tables allow the deleted key to be physically removed when it is marked deleted, as explained below.
- Use REORGCHK to help determine when to reorganize indexes or tables, or both, and when to use the REORG INDEXES with the CLEANUP ONLY option.

  To allow read and write access to the index during reorganization, run REORG INDEXES with the ALLOW WRITE ACCESS option.

Note: In DB2® Version 8.1 and later, all new indexes are created as type-2 indexes. The one exception is when you add an index on a table that already has type-1 indexes. In this case only, the new index will also be a type-1 index. To find out what type of index exists for a table, execute the INSPECT command. To convert type-1 indexes to type-2 indexes, execute the REORG INDEXES command.

The primary advantages of type-2 indexes are as follows:

- An index can be created on columns whose length is greater than 255 bytes.
- The use of next-key locking is reduced to a minimum, which improves concurrency. Most next-key locking is eliminated because a key is marked deleted instead of being physically removed from the index page. For information about key locking, refer to topics that discuss the performance implications of locks.

Index keys that are marked deleted are cleaned up in the following circumstances:

- During subsequent insert, update, or delete activity

  During key insertion, keys that are marked deleted and are known to be committed are cleaned up if such a cleanup might avoid the need to perform a page split and prevent the index from increasing in size.

During key deletion, when all keys on a page have been marked deleted an attempt is made to find another index page where all the keys are marked deleted and all those deletions have committed. If such a page is found, it is deleted from the index tree.

If there is an X lock on the table when a key is deleted, the key is physically deleted instead of just being marked deleted. During this physical deletion, any deleted keys on the same page are also removed if they are marked deleted and known to be committed.

- When you execute the REORG INDEXES command with CLEANUP options

  The CLEANUP ONLY PAGES option searches for and frees index pages on which all keys are marked deleted and known to be committed.

  The CLEANUP ONLY ALL option frees not only index pages on which all keys are marked deleted and known to be committed, but it also removes RIDs marked deleted and known to be committed on pages that contain some undeleted RIDs.

  This option also tries to merge adjacent leaf pages if doing so results in a merged leaf page that has at least PCTFREE free space on the merged leaf page. The PCTFREE value is the percent of free space defined for the index when it is created. The default PCTFREE is ten percent. If two pages can be merged, one of the pages will be freed.

- Any rebuild of an index

  Utilities that rebuild indexes include the following:

  – REORG INDEXES when not using one of the CLEANUP options
  – REORG TABLE when not using the INPLACE option
  – IMPORT with the REPLACE option
  – LOAD with the INDEXING MODE REBUILD option

Related concepts:

- “Advantages and disadvantages of indexes” on page 294
- “Index planning tips” on page 296
- “Index structure” on page 31
- “Table reorganization” on page 287
- “Index reorganization” on page 303

Index reorganization

As tables are updated with deletes and inserts, index performance degrades in the following ways:

- Fragmentation of leaf pages

  When leaf pages are fragmented, I/O costs increase because more leaf pages must be read to fetch table pages.

- The physical index page order no longer matches the sequence of keys on those pages, which is referred to as a *badly clustered* index.

  When leaf pages are badly clustered, sequential prefetching is inefficient and results in more I/O waits.
- The index develops more than its maximally efficient number of levels.

  In this case, the index should be reorganized.

If you set the MINPCTUSED parameter when you create an index, the database server automatically merges index leaf pages if a key is deleted and the free space is less than the specified percent. This process is called *online index defragmentation*. However, to restore index clustering, free space, and reduce leaf levels, you can use one of the following methods:

- Drop and recreate the index.
- Use the REORG INDEXES command to reorganize indexes online.

  You might choose this method in a production environment because it allows users to read from and write to the table while its indexes are being rebuilt.
- Use the REORG TABLE command with options that allow you to reorganize both the table and its indexes off-line.

Online index reorganization

When you use the REORG INDEXES command with the ALLOW WRITE ACCESS option, all indexes on the specified table are rebuilt while read and write access to the table is allowed. Changes made to the table during the reorganization process are applied after the indexes have been rebuilt. As a result, the rebuilt index might not be perfectly clustered. If PCTFREE is specified for an index, that percent of space is preserved on each page during reorganization.

Note: The CLEANUP ONLY option of the REORG INDEXES command does not fully reorganize indexes. The CLEANUP ONLY ALL option removes keys that are marked deleted and are known to be committed. It also frees pages in which all keys are marked deleted and are known to be committed. When pages are freed, adjacent leaf pages are merged if doing so can leave at least PCTFREE free space on the merged page. PCTFREE is the percentage of free space defined for the index when it is created. The CLEANUP ONLY PAGES option deletes only pages in which all keys are marked deleted and are known to be committed.

When you execute REORG INDEXES to reorganize an index, type-1 indexes are automatically converted to type-2 indexes, which use a method of deleting keys that avoids next-key locking and thus improves concurrency.

REORG INDEXES has the following requirements:

- SYSADM, SYSMAINT, SYSCTRL or DBADM authority, or CONTROL privilege on the indexes and table
- An amount of free space in the table space where the indexes are stored equal to the current size of the index

  Consider placing indexes subject to reorganization in a large table space when you issue the CREATE TABLE statement.
- Additional log space

  REORG INDEXES logs its activity. As a result, the reorganization might fail, especially if the system is busy and other concurrent activity is logged.

  Note: If a REORG INDEXES ALL with the ALLOW NO ACCESS option fails, the indexes are marked bad and the operation is not undone. However, if a REORG with the ALLOW READ ACCESS or a REORG with the ALLOW WRITE ACCESS option fails, the original index object is restored.

Related concepts:

- “Advantages and disadvantages of indexes” on page 294
- “Index planning tips” on page 296
- “Index performance tips” on page 299
- “Online index defragmentation” on page 305
- “Index cleanup and maintenance” on page 302

Related tasks:

- “Choosing a table reorganization method” on page 291

Online index defragmentation

Online index defragmentation is enabled by the user-definable threshold for the maximum amount of free space on an index leaf page. When an index key is deleted from a leaf page and the threshold is exceeded, 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 immediately in the background.

Online defragmentation of the index is only possible with indexes created in Version 6 and later. If existing indexes require the ability to be merged online, they must be dropped and then re-created with the `MINPCTUSED` clause. Set the `MINPCTUSED` value to less than one hundred (100). The recommended value for `MINPCTUSED` is less than 50 because the goal is to merge two neighboring index leaf pages. A value of zero for `MINPCTUSED`, which is also the default, disables online defragmentation.

Pages in the index are freed when the last index key on the page is removed. The exception to this rule occurs when you specify MINPCTUSED clause in the CREATE INDEX statement. The MINPCTUSED clause specifies a percent of space on an index leaf page. When an index key is deleted, if the percent of filled space on the page is at or below the specified value, then the database manager tries to merge the remaining keys with keys on an adjacent page. If there is sufficient space on an adjacent page, the merge is performed and an index leaf page is deleted.

Index non-leaf pages are not merged during an online index defragmentation. However, empty non-leaf pages are deleted and made available for re-use by other indexes on the same table. To free these non-leaf pages for other objects in a DMS storage model or to free disk space in an SMS storage model, perform a full reorganization of the table or indexes. Full reorganization of the table and indexes can make the index as small as possible. Index non-leaf pages are not merged during an online index defragmentation, but are deleted and freed for re-use if they become empty. The number of levels in the index and the number of leaf and non-leaf pages might be reduced.

For type-2 indexes, keys are removed from a page during key deletion only when there is an X lock on the table. During such an operation, online index defragmentation will be effective. However, if there is not an X lock on the table during key deletion, keys are marked deleted but are not physically removed from the index page. As a result, no defragmentation is attempted.

To defragment type-2 indexes in which keys are marked deleted but remain in the physical index page, execute the REORG INDEXES command with the CLEANUP ONLY ALL option. The CLEANUP ONLY ALL option defragments the index, regardless of the value of MINPCTUSED. If you execute REORG INDEXES with the CLEANUP ONLY ALL, two neighbouring leaf pages are merged if such a merge can leave at least PCTFREE free space on the merged page. PCTFREE is specified at index creation time and defaults to ten percent.

Related concepts:

- “Advantages and disadvantages of indexes” on page 294
- “Index performance tips” on page 299
- “Index structure” on page 31
- “Index reorganization” on page 303

DMS device considerations

If you use Database Managed Storage (DMS) device containers for table spaces, consider the following factors for effective administration:

- **File system caching**

  File system caching is performed as follows:

  – For DMS file containers (and all SMS containers), the operating system might 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: 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 usually available in the database buffer pool. In some cases, a data page might be freed from the buffer pool before the application has actually used the page, particularly if the buffer pool space is required for other data pages. For table spaces that use system managed storage (SMS) or database managed storage (DMS) file containers, file system caching above 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 might increase the size of the database buffer pool and reduce the size of the file system cache to offset the fact DMS table spaces that use device containers do not use double buffering.

  If system-level monitoring tools show that I/O is higher for a DMS table space using device containers compared to the equivalent SMS table space, this difference might be because of double buffering.

- **Using LOB or LONG data**

  When an application retrieves either LOB or LONG data, the database manager does not cache the data in its buffers, Each 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 might provide buffering and, as a result, better performance.

Because system catalogs contain some LOB columns, you should keep them in SMS table spaces or in DMS-file table spaces.

Related concepts:

- "Database directories and files" on page 16
- "SMS table spaces" on page 19
- "DMS table spaces" on page 20

Agent management

This section describes how the database manager uses agents and how to manage agents for good performance.

Database agents

For each database that an application accesses, various processes or threads start to perform the various application tasks. These tasks include logging, communication, and prefetching.

Database agents are engine dispatchable unit (EDU) processes or threads. Database agents do the work in the database manager that applications request. In UNIX® environments, these agents run as processes. In Intel-based operating systems such Windows, the agents run as threads.

The maximum number of application connections is controlled by the *max_connections* database manager configuration parameter. The work of each application connection is coordinated by a single worker agent.

A *worker agent* carries out application requests but has no permanent attachment to any particular application. The coordinator 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:

- Idle agents
- Inactive agents
- Active coordinator agents
- Subagents

Idle agents

This is the simplest form of worker agent. It does not have an outbound connection and it does not have a local database connection or an instance attachment.

Inactive agents

An inactive agent is a worker agent that is not in an active

transaction, does not have an outbound connection, and does not have a local database connection or an instance attachment. Inactive agents are free to begin doing work for an application connection.

Active coordinator agents

Each process or thread of a client application has a single active agent that coordinates its work on a database. After the coordinator agent is created, it performs all database requests on behalf of its application, and communicates to other agents using inter-process communication (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 become an inactive agent.

When a client disconnects from a database or detaches from an instance its coordinating agent will be:

- An active agent. If other connections are waiting, the worker agent becomes an active coordinator agent.
- Freed and marked as idle, if no connections are waiting and the maximum number of pool agents has not been reached.
- Terminated and its storage freed, if no connections are waiting and the maximum number of pool agents has been reached.

Subagents

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. After 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.

Agents that are not performing work for any applications and that 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 for client programs or for subagents operating for existing coordinator agents. The number of available agents depends on the database manager configuration parameters *maxagents* and *num_poolagents*.

When an agent finishes its work but still has a connection to a database, it is placed in the agent pool. Regardless of whether the connection concentrator is enabled for the database, if an agent is not waked up to serve a new request within a certain period of time and the current number of active and pooled agents is greater than *num_poolagents*, the agent is terminated.

Agents from the agent pool (*num_poolagents*) are re-used as coordinator agents for the following kinds of applications:

- Remote TCP/IP-based applications
- Local applications on UNIX-based operating systems
- Both local and remote applications on Windows® operating systems.

Other kinds of remote applications always create a new agent. If no idle agents exist when an agent is required, a new agent is created dynamically. Because creating a new agent requires a certain amount of overhead CONNECT and ATTACH performance is better if an idle agent can be activated for a client.

When a subagent is performing work for of an application, it is *associated* with that application. After it completes the assigned work, it can 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 before it creates a new agent.

Related concepts:

- "Database-agent management" on page 310
- "Agents in a partitioned database" on page 315
- "Connection-concentrator improvements for client connections" on page 312
- "Configuration parameters that affect the number of agents" on page 311

Database-agent management

Most applications establish a one-to-one relationship between the number of connected applications and the number of application requests that can be processed by 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.

The ability to control these factors separately is provided by two database manager configuration parameters:

- The *max_connections* parameter, which specifies the number of connected applications
- The *max_coordagents* parameter, which specifies the number of application requests that can be processed

The connection concentrator is enabled when the value of *max_connections* is greater than the value of *max_coordagents*.

Because each active coordinator agents requires global resource overhead, the greater the number of these agents the greater the chance that the upper limits of available database global resources will be reached. To prevent reaching the

upper limits of available database global resources, you might set the value of *max_connections* higher than the value of *max_coordagents*.

Related concepts:

- "Agents in a partitioned database" on page 315
- "Connection-concentrator improvements for client connections" on page 312
- "Configuration parameters that affect the number of agents" on page 311

Configuration parameters that affect the number of agents

The following database manager configuration parameters determine how many database agents are created and how they are managed:

- Maximum Number of Agents (*maxagents*): The number of agents that can be working at any one time. This value applies to the total number of agents that are working on all applications, including coordinator agents, subagents, inactive agents, and idle agents.
- Agent Pool Size (*num_poolagents*): The total number of agents, including active agents and agents in the agent pool, that are kept available in the system. The default value for this parameter is half the number specified for *maxagents*.
- Initial Number of Agents in Pool (*num_initagents*): 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 Connections (*max_connections*): specifies the maximum number of connections allowed to the database manager system on each partition.
- Maximum Number of Coordinating Agents (*max_coordagents*): For partitioned database environments and environments with intra-partition parallelism enabled when the connection coordinator is enabled, this value limits the number of coordinating agents.
- Maximum Number of Concurrent Agents (*maxcagents*): 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 from the database manager. This permission is called a *processing token*. The database manager permits only agents that have a processing token to execute a unit of work against a database. If a token is not available, the agent must wait until one is available to process the transaction.

  This parameter can be useful in an environment in which peak usage requirements exceed system resources for memory, CPU, and disk. For example, in such an environment, paging might cause performance

degradation for peak load periods. You can use this parameter to control the load and avoid performance degradation, although it can affect either concurrency or wait time, or both.

Related concepts:

- "Database agents" on page 308
- "Database-agent management" on page 310
- "Agents in a partitioned database" on page 315

Connection-concentrator improvements for client connections

For Internet applications with many relatively transient connections, or similar kinds of applications, the connection concentrator improves performance by allowing many more client connections to be processed efficiently. It also reduces memory use for each connection and decreases the number of context switches.

Note: The connection concentrator is enabled when the value of *max_connections* is greater than the value of *max_coordagents*.

In an environment that requires many simultaneous user connections, you can enable the connection concentrator for more efficient use of system resources. This feature incorporates advantages formerly found only in DB2Connect connection pooling. Both connection pooling and the connection concentrator are described in the DB2 Connect User's Guide. After the first connection, the connection concentrator reduces the connect time to a host. When a disconnection from a host is requested, the inbound connection is dropped, but the outbound connection to the host is kept in a pool. When a new request is made to connect to the host, DB2® tries to reuse an existing outbound connection from the pool.

Note: When applications use connection pooling or the connection concentrator, for best performance tune the parameters that control the size of the block of data that is cached. For more information, refer to the DB2 Connect User's Guide.

With DB2Connect connection pooling and the connection concentrator, the active agent does not close its outbound connection after a client disconnects, but is placed in the agent pool for the application, where it becomes a *logical subagent*, which is controlled by a *logical coordinator agent*. with an active connection to the remote host.

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. In connection pooling these idle agents are called *inactive agents*. The

pool of inactive agents is a synonym for the outbound connection pool. The connection concentrator implements a similar method of retaining inactive agents for later use in an application-specific pool.

Usage examples:

1. Consider an ESE environment with a single database partition in which, on average, 1000 users are connected to the database. At times, the number of concurrent transactions is as high as 200, but never higher than 250. Transactions are short.

   For this workload, the administrator sets the following database manager configuration parameters:

   - *max_connections* is set to 1000 to ensure support for the average number of connections.
   - *max_coordagents* is set to 250 to support the maximum number of concurrent transactions.
   - *maxagents* is set high enough to support all of the coordinator agents and subagents (where applicable) that are required to execute transactions on the node.

     If *intra_parallel* is OFF, *maxagents* is set to 250 because in such an environment, there are no subagents. If *intra_parallel* is ON, *maxagents* should be set large enough to accommodate the coordinator agent and the subagents required for each transaction that accesses data on the node. For example, if each transaction requires 4 subagents, *maxagents* should be set to (4+1) * 250, which is 1250. To tune *maxagents* further, take monitor snapshots for the database manager. The high-water mark of the agents will indicate the appropriate setting for *maxagents*.
   - *num_poolagents* is set to at least 250, or as high as 1250, depending on the value of *maxagents* to ensure that enough database agents are available to service incoming client requests without the overhead of creating new ones.

     However, this number could be lowered to reduce resource usage during low-usage periods. Setting this value too low causes agents to be deallocated instead of going into the agent pool, which requires new agents to be created before the server is able to handle an average workload.
   - *num_init_agents* is set to be the same as *num_poolagents* because you know the number of agents that should be active. This causes the database to create the appropriate number of agents when it starts instead of creating them before a given request can be handled.

   The ability of the underlying hardware to handle a given workload is not discussed here. If the underlying hardware cannot handle X-number of agents working at the same time, then you need to reduce this number to the maximum that the hardware can support. For example, if the

maximum is only 1500 agents, then this limits the maximum number of concurrent transactions that can be handled. You should monitor this kind of performance-related setting because it is not always possible to determine exact requests sent to other nodes at at given point in time.

2. In a system in which the workload needs to be restricted to a maximum 100 concurrent transactions and the same number of connected users as in example 1, you can set database manager configuration parameters as follows:
   - *max_coordagents* is set to 100
   - *num_poolagents* is set to 100

   With these settings, the maximum number of clients that can concurrently execute transactions is 100. When all clients disconnect, 100 agents are waiting to service new client connections. However, you should set *maxagents*, based on the type of workload, intra-query parallelism settings, the number of database partitions, and the underlying hardware.
3. Consider next an ESE installation with five database partitions, in which each partition has an average of 1000 user connections, and the concurrent transactions are as high as 200 but never higher than 250, set database configuration parameters as follows:
   - *max_coordagents* is set to 250 because, as in example 1, at most 250 clients execute transactions concurrently.
   - *maxagents* is set to 1500 Assuming data is distributed across the five partitions, each transaction may execute at least one subsection on each of the nodes in the system. ((1 coordinator agent + 5 subagents) * 250 = 1500.)
   - *num_poolagents* is set to 1200. (Assuming an average of 200 concurrent transaction with a maximum of 250. As a result, the number of agents required on average will be (1+5)*200 = 1200.)
   - *num_init_agents* is set to be the same as *num_poolagents*, as in example 1.
4. In a system for which you do not want to enable the connection concentrator but want to allow for 250 connected users at one time, set the database manager configuration parameters as follows:
   - *max_connections* is set to 250.
   - *max_coordagents* is set to 250.

Related concepts:

- “Database agents” on page 308
- “Database-agent management” on page 310
- “DB2 architecture and process overview” on page 11
- “Memory management” on page 44

Agents in a partitioned database

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.

Note: If the connection concentrator is enabled, subagents are not necessarily associated with an application.

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, which is also called a node. If the pool size is too small and the pool is full, a subagent disassociates itself from the application it is working on and terminates. Because subagents must be constantly created and re-associated to applications, performance suffers.

  In addition, if the value of *num_poolagents* is too small, one application may fill the pool with associated subagents. Then 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 costly, and causes poor performance.
- Weigh concerns about having too few agents 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, using database manager resources that are not available for other tasks.

  Note: When the connection concentrator is enabled, the number of agents specified by *num_poolagents* is only advisory. More agents might be in the agent pool at any given time.

Other asynchronous processes and threads

In addition to the database agents, other asynchronous database-manager activities run as their own process or thread including:

- Database I/O servers or I/O prefetchers
- Database asynchronous page cleaners

- Database loggers
- Database deadlock detectors
- Event monitors
- Communication and IPC listeners
- Table space container rebalancers.

Related concepts:

- "I/O server configuration for prefetching and parallelism" on page 279
- "Illustration of prefetching with parallel I/O" on page 280
- "Database agents" on page 308
- "Database-agent management" on page 310
- "Configuration parameters that affect the number of agents" on page 311

The database system-monitor information

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.)

You can access the data that the database manager maintains either by taking a snapshot or by using an event monitor.

Taking a snapshot

You can take a snapshot in one of the following three ways:

- Use the GET SNAPSHOT command from the command line.

- Use the Control Center on the Windows® operating systems for a graphical interface
- Write your own application, using the sqlmonss() API call.

Using the Control Center

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 lets you specify threshold values on performance variables to define exception conditions. When a threshold value is reached, one or more action that you define occurs: notification through a window or audible alarm or execution of a script or program.

If you take a snapshot from the Control Center, while you are performing snapshot monitoring on either that object, or on any it its child objects you cannot perform an action that either alters, changes, or deletes a database object, such as an instance or database. 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 a Windows-based operating system 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.

Related concepts:

• "Quick-start tips for performance tuning" on page 7

Chapter 9. Using the governor

This chapter describes how to set up and run the governor tool to monitor and control database activity.

The Governor utility

The governor can monitor the behavior of applications that run against a database and can change certain behavior, depending on the rules that you specify in the governor configuration file.

A governor instance consists of a front-end utility and one or more daemons. Each instance of the governor that you start is specific to an instance of the database manager. By default, when you start the governor a governor daemon starts on each partition of a partitioned database. However, you can specify that a daemon be started on a single partition that you want to monitor.

Note: When the governor is active, its snapshot requests might affect database manager performance. To improve performance, increase the governor wake-up interval to reduce its CPU usage.

Each governor daemon collects information about the applications that run against the database. If then checks this information against the rules that you specify in the governor configuration file for this database.

The governor manages application transactions as specified by the rules in the configuration file. For example, applying a rule might indicate that an application is using too much of a particular resource. The rule would specify the action to take, such as to change the priority of the application or force it to disconnect from the database.

If the action associated with a rule changes the priority of the application, the governor changes the priority of agents on the database partition where the resource violation occurred. In a partitioned database, if the application is forced to disconnect from the database, the action occurs even if the daemon that detected the violation is running on the coordinator node of the application.

The governor logs any actions that it takes. To review the actions, you query the log files.

Related concepts:

- "The Governor daemon" on page 321
- "The Governor configuration file" on page 324
- "Governor log files" on page 333

Related tasks:

- "Starting and stopping the governor" on page 320
- "Configuring the Governor" on page 323

Related reference:

- "db2gov - DB2 Governor Command" in the *Command Reference*

Governor startup and shutdown

This section explains how to start and stop the governor tool and describes the activity of the governor daemon.

Starting and stopping the governor

The governor utility monitors applications that connect to a database and changes their behavior according to rules that you specify in a governor configuration file for that database.

Prerequisites:

Before you start the governor, you must create the configuration file.

Restrictions:

To start or stop the governor, you must have *sysadm* or *sysctrl* authorization.

Procedure:

To start or stop the governor:

1. To start the governor, execute the *db2gov* command at the DB2 command line. Enter the following required parameters:
   - START *database_name*

     The database name that you specify must match the name of the database in the configuration file that you specify. An error is returned if the names are not the same. Note that if a governor is running for more than one database, daemons will be started for each database.
   - *config_file_name*

The name of the configuration file for the governor on this database. If the file is not in the default location, which is the `sqllib` directory, you must include the path as well as the file name.

- *log_file_name*

  The base name of the log file for this governor. On a partitioned database, the partition number is added for each partition where a daemon runs for this instance of the governor.

To start the governor on a single partition for a partitioned database, add the *nodenum* option. For example, to start the governor for a database called *sales* on only node 3 of a partitioned database with a configuration file called *salescfg* and a log file called *saleslog*, enter the following command:

```
db2gov START sales nodenum 3 salescfg saleslog
```

To start the governor on all partitions of the *sales* database, enter the following command:

```
db2gov START sales  salescfg saleslog
```

2. To stop the governor, enter the *db2gov* command with the STOP option.

   For example, to stop the governor on all partitions of the *sales* database, enter the following command:

   ```
   db2gov STOP sales
   ```

   To stop the governor on only partition 3, enter the following command:

   ```
   db2gov START sales nodenum 3
   ```

Related concepts:

- "The Governor utility" on page 319
- "The Governor daemon" on page 321

Related reference:

- "db2gov - DB2 Governor Command" in the *Command Reference*

The Governor daemon

When the governor daemon starts, either when you execute by the `db2gov` utility or when it wakes up, it runs the following task loop.

1. It checks 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.
2. It requests snapshot information about resource-use statistics for each application and agent that is working on the database.

Note: On some platforms, the CPU statistics are not available from the DB2® Monitor. In this case, the account rule and the CPU limit are not available.

3. It checks the statistics for each application against the rules in the governor configuration file. If a rule applies to an application, the governor performs the specified action.
4. It writes a record in the governor log file for any action that it takes.

Note: The governor cannot be used to adjust agent priorities if the *agentpri* database manager configuration parameter is anything other than the system default. (This note does not apply to Windows® NT platforms.)

When the governor finishes its tasks, it sleeps for the interval specified in the configuration file. when the interval elapses, the governor wakes up and begins the task loop again.

When the governor encounters an error or stop signal, it does cleanup processing before it ends. Using a list of applications whose priorities have been set, the cleanup processing resets all application agent priorities. 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, the governor writes a message to the administration notification log to indicate that it ended abnormally.

Note: Although the governor daemon is not a database application, and therefore does not maintain a connection to the database, it does have an instance attachment. Because it can issue snapshot requests, the governor daemon can detect when the database manager ends.

Related concepts:

• "The Governor utility" on page 319

Related tasks:

• "Starting and stopping the governor" on page 320

Governor configuration

This section explains how to configure the governor to monitor and control database activity.

Configuring the Governor

To configure the Governor, you create a configuration file that determines the database that an instance of the Governor monitors and how it manages queries.

The configuration file consists of a set of rules. The first three rules specify the database to monitor, the interval at which to write log records, and the interval at which to wake up for monitoring. The remaining rules specify how to monitor the database server and what actions to take in specific circumstances.

Procedure:

To create a Governor configuration file:

1. In a directory that is mounted or available from all database manager partitions, create an ASCII file with a descriptive name. For example, the configuration file for a governor instance that monitors the **sales** database might be called *govcfgsales*.
2. Open the file in any text editor and enter configuration information and action conditions.

   End each rule with a semicolon (;). The following configuration information is recommended:

   - dbname: The name or alias of the database to be monitored.
   - account: The number of minutes after which the governor instance writes CPU usage statistics to its log file. This option is not available on Windows NT.
   - interval: The number of seconds after which the governor daemon wakes up to monitor activity. If you do not specify an interval, the default value of 120 seconds is used.

   For example, the first three rules in the configuration file might look like this:

   ```
   { Wake up once a second, the database name is sales,
    do accounting every 30 minutes.  }
   interval 1; dbname sales; account 30;
   ```

   Add rules that specify the conditions to monitor and the action to take if the rule evaluates to true. For example, you might add a rule that limits to an hour the amount of time that a unit of work (UOW) can run before being forced to disconnect from the database, as follows:

   ```
   setlimit uowtime 3600 action force;
   ```

3. Save the file.

Related concepts:

- "The Governor configuration file" on page 324
- "Governor rule elements" on page 326
- "Example of a Governor configuration file" on page 331
- "Governor log files" on page 333

Related reference:

- "db2gov - DB2 Governor Command" in the *Command Reference*

The Governor configuration file

When you start the governor, you specify the configuration file that contains the rules that govern applications running against the database. The governor evaluates each rule and acts as specified when the rule evaluates to true.

If your rule requirements change, you edit the configuration file without stopping the governor. Each governor daemon detects that the file has changed, and reread it.

The configuration file must be created in a directory that is mounted across all the database partitions so that the governor daemon on each partition can read the same configuration file.

The configuration file consists of three required rules that identify the database to be monitored, the interval at which log records are written, and the sleep interval of the governor daemons. Following these parameters, the configuration file contains a set of optional application-monitoring rules and actions. The following comments apply to all rules:

- Delimit comments inside { } braces.
- Most entries can be specified in uppercase, lowercase, or mixed case characters. The exception is the application name, specified as an argument to the `applname` rule, which is case sensitive.
- Each rule ends with a semicolon (;).

Required rules

The following rules specify the database to be monitored and the interval at which the daemon wakes up after each loop of activities. Each of these rules is specified only 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 the Windows NT environment.

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.

interval

The interval, in seconds, at which the daemon wakes up. If you do not specify an interval, the default value, 120 seconds, is used.

Rules that govern actions

Following the required rules, you can add rules that specify how to govern the applications. These rules are made of smaller components called rule clauses. If used, the clauses must be entered in a specific order in the rule statement, as follows:

1. **desc** (optional): a comment about the rule, enclosed in quotation marks
2. **time** (optional): the time during the day when the rule is evaluated
3. **authid** (optional): one or more authorization IDs under which the application executes statements
4. **applname** (optional): the name of the executable or object file that connects to the database. This name is case sensitive.
5. **setlimit**: the limits that the governor checks. These can be one of several, including CPU time, number of rows returned, idle time, and so on.
6. **action** (optional): the action to take if a limit is reached. If no action is specified, the governor reduces the priority of agents working for the application by 10 when a limit is reached. Actions against the application can include reducing its agent priority, forcing it to disconnect from the database, or setting scheduling options for its operations.

You combine the rule clauses to form a rule, using each clause only once in each rule, and end the rule with a semicolon, as shown in the following examples:

```
desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;
```

```
desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;
```

If more than one rule applies to an application, all are applied. Usually, the action associated with the rule limit encountered first is the action that is applied first. An exception occurs you specify if -1 for a clause in a rule. In this case, 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 uses the `rowssel 100000 uowtime 3600` clauses to specify that the priority of an application is decreased either if its elapsed time is greater than 1 hour or if it selects more than 100 000 rows. A subsequent rule uses the `uowtime -1` clause to specify that the same application can have unlimited elapsed time. In this case, if the application runs for more than 1 hour, its priority is not changed. That is, `uowtime -1` overrides `uowtime 3600`. However, if it selects more than 100 000 rows, its priority is lowered because `rowssel 100000` is still valid.

Related concepts:

- "Governor rule elements" on page 326
- "Example of a Governor configuration file" on page 331

Governor rule elements

Each rule in the governor configuration file is made up of clauses that specify the conditions for applying the rule and the action that results if the rule evaluates to true. The clauses must be specified in the order shown. In the clause descriptions, [ ] indicates an optional clause.

Optional beginning elements

[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.

Limit clauses

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 the Windows NT environment.

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 clauses

[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 Windows® 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 chooses the top applications for scheduling based on the following three criteria:

- The application holding the most locks

  This choice is an attempt to reduce the number of lockwaits.
- The oldest application
- The application with the shortest estimated remaining running time.

  This choice is an attempt to allow as many short-lived statements as possible to complete during the interval.

The top three applications in each criteria are given higher priorities than all other applications That is, the top application in each criterion group is given the highest priority, the next highest applications are given the second highest priority and the third-ranked applications are given the third highest priority. If a single application is ranked in the top three in more than one of the criteria, it is given the appropriate priority for the criterion in which it ranked highest, and the next highest application is given the next highest priority for the other criteria. For example, if application A holds the most locks but has the third shortest estimated remaining running time, it is given the highest priority for the first criterion, and the fourth ranked application with the shortest estimated remaining running time is given the third highest priority for that criterion.

The applications selected by this governor rule are divided in up to three classes. For each class, the governor chooses nine applications, which are the top three applications from each class, based on the criteria listed above. If you specify the class option, all applications selected by this rule are

considered a single class, and nine applications are chosen and given higher priorities as described above.

If an application is selected in more than one governor rule, it is governed by the last rule in which is it selected.

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 14 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 active 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.

Related concepts:

- "The Governor configuration file" on page 324
- "Example of a Governor configuration file" on page 331

Related tasks:

- "Configuring the Governor" on page 323

Example of a Governor configuration file

The following example shows a governor configuration file that sets several rules with actions:

```
{ 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;
```

Related concepts:

- "The Governor configuration file" on page 324
- "Governor rule elements" on page 326

Related tasks:

- "Configuring the Governor" on page 323

Governor log-file use

This section describes the governor log files and explains how to query them to retrieve information.

Governor log files

Whenever a governor daemon performs an action, it writes a record to its log file. Actions include the following:

- Forcing an application
- Reading the governor configuration file
- Changing an application priority
- Encountering an error or warning
- Starting or ending

Each governor daemon has a separate log file. Separate log files prevents file-locking bottlenecks that might result when many governor daemons write to the same file at the same time. To merge the log files together and query them, use the `db2govlg` utility.

The log files are stored in the `log` subdirectory of the `sqllib` directory, except on Windows® NT, where the `log` subdirectory is under the instance directory. You provide the base name for the log file when you start the governor with the `db2gov` command. Make sure that the log file name contains the database name to distinguish log files on each partition of each that is governed. To ensure that the filename is unique for each governor in a partitioned database environment, the partition number where the governor daemon runs is automatically appended to the log file name.

Log file record format

Each record in the log file has the following format:

```
Date Time NodeNum RecType Message
```

Note: The format of the *Date* and *Time* fields is yyyy-mm-dd hh.mm.ss. 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: the governor was started
- FORCE: an application was forced
- PRIORITY: the priority of an application was changed
- ERROR: an error occurred
- WARNING: a warning occurred
- READCFG: the governor read the configuration file
- STOP: the governor was stopped
- ACCOUNT: the application accounting statistics.

  The accounting statistics fields are:

  – authid
  – appl_id
  – written_usr_cpu
  – written_sys_cpu
  – appl_con_time
- SCHEDULE: 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
```

Related concepts:

- "The Governor utility" on page 319
- "Governor log-file queries" on page 334

Governor log-file queries

Each governor daemon writes to its own log file. You can use db2govlg utility to query the log file. You can list the log files for a single partition, or for all database partitions, sorted by date and time. You can also query on the basis of the *RecType* log field. The syntax for db2govlg is as follows:

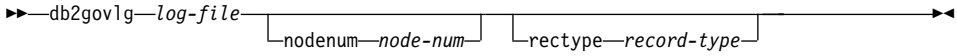


Figure 25. Syntax for db2govlg

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.

Related concepts:

- "The Governor utility" on page 319
- "Governor log files" on page 333

Chapter 10. Scaling your configuration

This chapter describes how you can manage database capacity, primarily by adding and dropping database partitions. Other methods of increasing capacity include adding CPUs and adding memory.

Management of database server capacity

If database manager capacity does not meet your present or future needs, you can expand its capacity in the following ways:

- Add disk space and create additional containers.
- Add memory.

If these simple strategies do not add the capacity you need, consider the following methods:

- Add processors.

  If a single-partition configuration with a single processor is used to its maximum capacity, you might either add processors or add partitions. The advantage of adding processors is greater processing power. In an SMP system, processors share memory and storage system resources. All of the processors are in one system, so there are no additional overhead considerations such as communication between systems and coordination of tasks between systems. Utilities in DB2® such as load, backup, and restore can take advantage of the additional processors. DB2 Universal Database™ supports this environment.

  Note: Some operating systems, such as Solaris, can dynamically turn processors on- and off-line.

  If you add processors, review and modify some database configuration parameters that determine the number of processors used. The following database configuration parameters determine the number of processors used and might need to be updated:

  – Default degree (dft_degree)
  – Maximum degree of parallelism (max_querydegree)
  – Enable intra-partition parallelism (intra_parallel)

  You should also evaluate parameters that determine how applications perform parallel processing.

In an environment where TCP/IP is used for communication, review the value for the DB2TCPCONNMGRS registry variable.

- Add physical nodes.

  If your database manager is currently partitioned, you can increase both data-storage space and processing power by adding separate single-processor or multiple-processor physical nodes. The memory and storage system resources on each node are not shared with the other nodes. Although adding nodes might result in communication and task-coordination issues, this choice provides the advantage of balancing data and user access across more than one system. DB2 Universal Database supports this environment.

  You can add nodes either while the database manager system is running or while it is stopped. If you add nodes while the system is running, however, you must stop and restart the system before databases migrate to the new node.

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.

Related concepts:

- "Partitions in a partitioned database" on page 338

Partitions in a partitioned database

You can add database partitions to the partitioned database system either when it is running, or when it is stopped. Because adding a new server can be time consuming, you may want to do it when the database manager is already running.

Use the ADD DBPARTITIONNUM command to add a database partition to a system. This command can be invoked in the following ways:

- As an option on `db2start`
- With the command-line processor ADD DBPARTITIONNUM command
- With the API function `sqleaddn`
- With the API function `sqlepstart`

If your system is stopped, you use `db2start`. If it is running, you can use any of the other choices.

When you use the ADD DBPARTITIONNUM command to add a new database partition to the system, all existing databases in the instance are expanded to the new database partition. You can also specify which containers to use for temporary table spaces for the databases. 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. You must use the ALTER TABLESPACE statement to add temporary table space containers to each database before the database can be used.

You cannot use a database on the new partition to contain data until one or more database partition groups are altered to include the new database partition.

Note: If no databases are 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 procedures described, as they apply only when a database exists.

Windows NT Considerations: If you are using DB2 Enterprise - Extended Edition on Windows NT and have no databases in the instance, use the DB2NCRT command to scale the database system. If, however, you already have databases, use the DB2START ADDNODE command to ensure that a database partition is created for each existing database when you scale the system. On Windows NT, you should never manually edit the node configuration file (`db2nodes.cfg`), as this can introduce inconsistencies into the file.

Related tasks:

- "Adding a partition to a running database system" on page 339
- "Adding a partition to a stopped database system on Windows NT" on page 341
- "Dropping a database partition" on page 346

Adding a partition to a running database 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.

Procedure:

To add a database partition to a running database manager:

1. On any existing database partition, run the DB2START command.

   On all platforms, specify the new partition values for DBPARTITIONNUM, ADD DBPARTITIONNUM, HOSTNAME, PORT, and NETNAME parameters. On the Windows NT platform, you also specify the COMPUTER, USER, and PASSWORD parameters.

   You can also specify the source for any temporary table space container definitions that need to be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog node for each database.

   When the DB2START command is complete, the new server is stopped.
2. Stop the database manager on all partitions 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. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DBPARTITIONNUM command, which is called when you specify the ADDNODE parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
3. 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, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
4. Back up all databases on the new database partition. (Optional)
5. Redistribute data to the new database partition. (Optional)

Related concepts:

- "Partitions in a partitioned database" on page 338

Related tasks:

- "Adding a partition to a stopped database system on Windows NT" on page 341
- "Adding a partition to a stopped database system on UNIX" on page 342

Adding a partition to a stopped database system on Windows NT

You can add new database partitions to a partitioned database system while it is stopped. The newly added database partition becomes available to all databases when the database manager is started up again.

Prerequisites:

You must install the new server before you can create a partition on it.

Procedure:

To add a partition to a stopped partitioned database server:

1. Issue DB2STOP to stop all the database partitions.
2. Run the ADD DBPARTITIONNUM command on the new server.

   A database partition is 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. Update the database configuration parameter values to match those on the other database partitions.
3. Run the DB2START command to start the database system. Note that the node configuration file (cfg) has already been updated to include the new server during the installation of the new server.
4. Update the configuration file on the new partition as follows:
   a. On any existing database partition, run the DB2START command.

      Specify the new partition values for DBPARTITIONNUM, ADDDB2PARTITIONNUM, HOSTNAME, PORT, and NETNAME parameters as well as the COMPUTER, USER, and PASSWORD parameters.

      You can also specify the source for any temporary table-space container definitions that need to be created with the databases. If you do not provide table-space information, temporary table-space container definitions are retrieved from the catalog node for each database.

      When the DB2START command is complete, the new server is stopped.

   b. Stop the entire 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. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DB2PARTITIONNUM command, which is called when you specify the ADDDB2PARTITIONNUM parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.

5. Start the database manager by running the DB2START command.

   The newly added database partition is now started with the rest of the system.

   When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
6. Back up all databases on the new database partition. (Optional)
7. Redistribute data to the new database partition. (Optional)

Related concepts:

- "Partitions in a partitioned database" on page 338
- "Node-addition error recovery" on page 344

Related tasks:

- "Adding a partition to a running database system" on page 339
- "Adding a partition to a stopped database system on UNIX" on page 342

Adding a partition to a stopped database system on UNIX

You can add new database partitions to a partitioned database system while it is stopped. The newly added database partition becomes available to all databases when the database manager is started up again.

Prerequisites:

You must install the new server if it does not exist, including the following tasks:

- 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
- Ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values

You must also register the host name with the name server or in the hosts file in the etc directory on all database partitions.

Procedure:

To add a partition to a stopped partitioned database server:

1. Issue DB2STOP to stop all the database partitions.
2. Run the ADD DB2PARTITIONNUM command on the new server.

   A database partition is 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. Update the database configuration parameter values to match those on the other database partitions.
3. Run the DB2START command to start the database system. Note that the node configuration file (cfg) has already been updated to include the new server during the installation of the new server.
4. Update the configuration file on the new partition as follows:
   a. On any existing database partition, run the DB2START command.

      Specify the new partition values for DB2PARTITIONNUM, ADDDB2PARTITIONNUM, HOSTNAME, PORT, and NETNAME parameters as well as the COMPUTER, USER, and PASSWORD parameters.

      You can also specify the source for any temporary table-space container definitions that need to be created with the databases. If you do not provide table-space information, temporary table-space container definitions are retrieved from the catalog node for each database.

      When the DB2START command is complete, the new server is stopped.
   b. Stop the entire 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. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DB2PARTITIONNUM command, which is called when you specify the ADDDB2PARTITIONNUM parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
5. Start the database manager by running the DB2START command.

   The newly added database partition is now started with the rest of the system.

   When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
6. Back up all databases on the new database partition. (Optional)
7. Redistribute data to the new database partition. (Optional)

You can also update the configuration file manually, as follows:

1. 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 DB2PARTITIONNUM partitionnum`

   Specify the number you are assigning to the new database partition 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 DB2PARTITIONNUM registry variable. Specify the number of the database partition you are adding.
4. Run the ADD NODE command on the new database partition.

   This command creates a database partition 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. Update the database configuration parameter values to match those on the other database partitions.
5. When the ADD DB2PARTITIONNUM command completes, issue the DB2START command to start the other database partitions in the system.

   Do not perform any system-wide activities, such as creating or dropping a database, until all database partitions are successfully started.

Related concepts:

- "Node-addition error recovery" on page 344

Related tasks:

- "Adding a partition to a running database system" on page 339
- "Adding a partition to a stopped database system on Windows NT" on page 341
- "Dropping a database partition" on page 346

Node-addition error recovery

Because in version 8.1 and later, DB2 creates "hidden" buffer pools to provide default automatic support for all buffer-pool page sizes, node-addition does not fail because of non-existent buffer pools. However, if one of these "hidden" buffer pools is used, performance might be seriously affected because the hidden buffer pools are very small. If a hidden buffer pool is used, a message is written to the administration notification log.

Hidden buffer pools are used in node-addition scenarios in the following circumstances:

- You add nodes to a partitioned database that has one or more system temporary table spaces with a page size that is different from the default of

4 KB. When a node is created, only the IBMDEFAULTDP buffer pool exists, and this buffer pool has a page size of 4 KB.

Consider the following examples:

1. You use the db2start command to add a node to the current partitioned database:

```
DB2START DB2PARTITIONNUM 2 ADD DB2PARTITIONNUM HOSTNAME newhost PORT 2
```

2. You use the ADD DB2PARTITIONNUM command after you manually update the db2nodes.cfg file with the new node description.

One way to prevent these problems is to specify the WITHOUT TABLESPACES clause on the ADD NODE or the **db2start** command. After doing this, you need to use the CREATE BUFFERPOOL statement to create the buffer pools using , and associate the system temporary table spaces to the buffer pool using the ALTER TABLESPACE statement.

- You add nodes to an existing database partition group that has one or more table spaces with a page size that is different from the default page size, which is 4 KB. This occurs because the non-default page-size buffer pools created on the new node have not been activated for the table spaces.

  Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

  Consider the following example:

  – You use the ALTER DATABASE PARTITION GROUP statement to add a node to a database partition group, as follows:

```
DB2START
CONNECT TO mpp1
ALTER DATABASE PARTITION GROUP ng1 ADD NODE (2)
```

  One way to prevent this problem is to create buffer pools for each page size and then to reconnect to the database before issuing the following ALTER DATABASE PARTITION GROUP statement:

```
DB2START
CONNECT TO mpp1
CREATE BUFFERPOOL bp1 SIZE 1000 PAGESIZE 8192
CONNECT RESET
CONNECT TO mpp1
ALTER DATABASE PARTITION GROUP ng1 ADD NODE (2)
```

Note: If the database partition group has table spaces with the default page size, the following message is returned:

```
SQL1759W Redistribute nodegroup is required to change data
      positioning for objects in nodegroup "ng1" to include some
      added nodes or exclude some drop nodes.
```

Related tasks:

- "Adding a partition to a running database system" on page 339
- "Adding a partition to a stopped database system on Windows NT" on page 341

Dropping a database partition

You can drop a database partition that is not being used by any database and free the computer for other uses.

Prerequisites:

Verify that the partition is not in use by issuing the DROP NODE VERIFY command or the *sqledrpn* API.

- If you receive message SQL6034W (Node not used in any database), you can drop the partition.
- If you receive message SQL6035W (Node in use by database), use the REDISTRIBUTE NODEGROUP command to redistribute the data from the database partition that you are dropping to other database partitions from the database alias.

Also 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 in-doubt transactions.

Procedure:

To drop a database partition:

1. Issue the DB2STOP command with the DROP NODENUM parameter to drop the database partition. After the command completes successfully, the system is stopped.
2. Start the database manager with the DB2START command.

Related concepts:

- "Management of database server capacity" on page 337
- "Partitions in a partitioned database" on page 338

Chapter 11. Redistributing Data Across Database Partitions

This chapter provides information about determining when to redistribute data across partitions, how to perform the redistribution, and how to recover from redistribution errors.

Data redistribution

To redistribute table data among the partitions in a partitioned database, you use the REDISTRIBUTE DATABASE PARTITION GROUP command.

Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

In a partitioned database you might redistribute data for the following reasons:

- To balance data volumes and processing loads across database partitions.

  Performance improves if data access can be spread out over more than one partition.
- To introduce skew in the data distribution across database partitions.

  Access and throughput performance might improve if you redistribute data in a frequently accessed table so that infrequently accessed data is on a small number of database partitions in the database partitioning group, 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.

To preserve table collocation, use the REDISTRIBUTE DATABASE PARTITION GROUP command to redistribute data at the database partitioning group level. All tables are redistributed in a single operation. To achieve a specified data distribution, the REDISTRIBUTE DATABASE PARTITION GROUP command divides tables among the database partitions as it moves the rows. Depending on the option you specify, the utility can either generate a target partitioning map or use an existing partitioning map as input.

How data is redistributed across database partitions

Data redistribution is performed on the set of tables in the specified database partitioning group of a database. You must connect to the database at the catalog database partition before executing REDISTRIBUTE DATABASE PARTITION GROUP command to invoke the Data Redistribution utility. 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, which 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 performs the following steps:

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.DBPARTITIONGROUPS catalog view for the database partitioning group with the new partitioning map ID.
3. Adds any new database partitions to the SYSCAT.DBPARTITIONGROUPDEF catalog view.
4. Sets the IN_USE column in the SYSCAT.DBPARTITIONGROUPDEF 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 database partitioning group, in the following steps:
   a. Locks the row for the table in the SYSTABLES catalog table.
   b. Invalidates all packages that involve this table. The partitioning map ID associated with the table changes because the table rows are redistributed. Because the packages are invalidated, the compiler must obtain the new partitioning information for the table and generate packages accordingly.
   c. Locks the table in exclusive mode.
   d. Uses DELETEs and INSERTs to redistribute the data in the table.
   e. If the redistribution operation succeeds, it issues a COMMIT for the table and continues with the next table in the database partitioning group. If the operation fails before the table is fully redistributed, the utility Issues a ROLLBACK on updates to the table, ends the entire redistribution operation and returns an error.
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 database partitioning group 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.

Related concepts:

- "Log space requirements for data redistribution" on page 352
- "Redistribution-error recovery" on page 353

Related tasks:

- "Redistributing data across partitions" on page 350
- "Determining whether to redistribute data" on page 349

Determining whether to redistribute data

Before you decide to repartition data, find out whether data is distributed unequally among partitions. After you have current distribution information, you can use this information to create a custom redistribution file or partitioning map.

Procedure:

To get information about current data distributions for partitions in a database partition group:

1. Determine if any database partitions have unequal distributions of rows.

   For the largest table, use an appropriate partitioning column and enter a query such as the following:

   ```
   SELECT PARTITION(column_name), COUNT(*) FROM table_name
      GROUP BY PARTITION(column_name)
      ORDER BY PARTITION(column_name) DESC
      FETCH FIRST 100 ROWS ONLY
   ```

   The PARTITION and DBPARTITIONNUM SQL functions determine the current data distribution across hash partitions or database partitions. The PARTITION function returns the partitioning map index for each row of the table. The DBPARTITIONNUM function returns the partition number of the row.
2. Execute this query for other large tables that are partitioned across the database partition group.
3. Use the information to create both a distribution file and a target partitioning map.

Note: You can also use `AutoLoader` utility with its ANALYZE option to create a data distribution file. You can use this file as input to the Data Redistribution utility.

Related concepts:

- “Data redistribution” on page 347
- “Log space requirements for data redistribution” on page 352

Related tasks:

- “Redistributing data across partitions” on page 350

Redistributing data across partitions

In a partitioned database, you might redistribute data among partitions to balance data access in the following cases:

- When some partitions contain more data than others
- When some partitions are accessed more frequently than others

Prerequisites:

Log file size: Ensure that log files are large enough for the data redistribution operation. The log file on each affected partition must be large enough to accommodate the INSERT and DELETE operations performed there.

Replicated materialized query tables: If the data in a database partition group contains replicated materialized query tables, you must drop these tables before you redistribute the data. After data is redistributed, you can recreate the materialized query tables.

Restrictions:

You can do the following operations on objects of the database partition group 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 database partition group. The CREATE TABLE statement uses the target partitioning map.
- Create table spaces in the database partition group.

You cannot do the following operations while the utility is running:

- Start another redistribution operation on the database partition group
- Execute an ALTER TABLE statement on any table in the database partition group
- Drop the database partition group
- Alter the database partition group.

Procedure:

To redistribute data across partitions in a database partition group:

1. Connect to the database partition that contains the system catalog tables.
2. Perform prerequisite tasks, if necessary.
3. Issue the REDISTRIBUTE DATABASE PARTITION GROUP command.

   Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

   Specify the following arguments:

   database partition group name
   You must specify the database partition group within which data is to be redistributed.

   UNIFORM
   If data is evenly distributed and is to remain evenly distributed, either specify UNIFORM or omit any distribution-type argument. UNIFORM is the default.

   USING DISTFILE distfile-name
   To specify a custom distribution that corrects or creates data skew, include the distribution file name. The Redistribute Data utility uses this file to construct a target partitioning map.

   USING TARGETMAP targetmap-name
   The Redistribute Data utility uses the specified target map directly.

   For details, refer to the REDISTRIBUTE DATABASE PARTITION GROUP command-line utility information.
4. After redistribution is complete:
   - Recreate any replicated materialized query tables dropped before redistribution.
   - Execute the RUNSTATS command to collect data distribution statistics for the SQL compiler and optimizer to use when it chooses data access plans for queries.

Note: The Explain tables contain information about the partitioning map used to redistribute data.

Related concepts:

- "Data redistribution" on page 347
- "Log space requirements for data redistribution" on page 352
- "Redistribution-error recovery" on page 353

Related tasks:

- "Determining whether to redistribute data" on page 349

Log space requirements for data redistribution

Before you redistribute data across partitions, consider the log-space requirements.

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, use the ratio of current database partitions to the new number of database partitions to estimate the number of INSERT and DELETE operations. For example, consider redistributing data that is uniformly distributed before redistribution. If you are moving from four to five database partitions, approximately twenty percent of the four original database partitions will move to the new database partition. This means that twenty percent of the DELETE operations will occur on each of the four original database partitions, and all of the INSERT operations will occur on the new database partition.

Consider a non-uniform distribution of the data, such as the case in which the partitioning key contains many NULL values. In this case, all rows that contain a NULL value in the partitioning key move from one database partition under the old partitioning scheme and to a different database partition under the new partitioning scheme. As a result, the amount of log space required on those two database partitions increases, perhaps well beyond the amount calculated by assuming uniform distribution.

The redistribution of each table is a single transaction. For this reason, when you estimate log space, you multiply the percentage of change, such as twenty percent, by the size of the largest table. Consider, however, that the largest table might be uniformly distributed but the second largest table, for example, might have one or more inflated database partitions. In such a case, consider using the non-uniformly distributed table instead of the largest one.

Note: After you estimate the maximum amount of data to be inserted and deleted at a database partition, double that estimate to determine the peak size of the active log. If this estimate is greater than the active log limit of 256 GB, then the data redistribution must be done in steps. Use the "makepmap" utility to generate a series of target partition maps, one for each step. You might also set the *logsecond* database configuration parameter to -1 to avoid most log space problems.

Related concepts:

• "Data redistribution" on page 347

Redistribution-error recovery

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:

```
For UNIX® platforms:
  databasename.database partition groupname.timestamp
For non-UNIX platforms:
  databasename\database partition groupname\date\time
```

Note: On non-UNIX platforms, only the first eight (8) bytes of the database partition-group name 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 SYSCAT.DBPARTITIONGROUPS 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.

Related concepts:

- "Data redistribution" on page 347

Related tasks:

- "Redistributing data across partitions" on page 350

Chapter 12. Benchmark testing

This chapter explains the benchmarking process and how to use the *db2batch* utility to perform benchmark testing of a database workload.

Benchmark testing

Benchmark testing is a normal part of the application development life cycle. It is a team effort that involves both application developers and database administrators (DBAs), and should be performed against your application in order to determine current performance and improve it. If the application code has been written as efficiently as possible, additional performance gains might be realized from tuning the database and database manager configuration parameters. You can even tune application parameters to meet the requirements of the application better.

You run different types of benchmark tests to discover specific kinds of information:

- A *transaction per second* benchmark determines the throughput capabilities of the database manager under certain limited laboratory conditions.
- An *application* benchmark tests the same throughput capabilities under conditions that are closer production conditions.

Benchmarking tuning configuration parameters is based upon these "real-world" conditions, and requires repeatedly running SQL taken from your application with varying parameter values until the application runs as efficiently as possible.

The benchmarking methods described here are oriented toward tuning 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 can 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.

Benchmark testing methods

Benchmark tests are based on a repeatable environment so that the same test run under the same conditions will yield results that you can legitimately compare.

You might begin benchmarking by running the test application in a normal environment. As you narrow down a performance problem, you can develop specialized test cases that limit the scope of the function that you are testing. The specialized test cases need not emulate an entire application to obtain valuable information. Start with simple measurements, and increase the complexity only when necessary.

Characteristics of good benchmarks or measurements include:

- Tests are repeatable.
- Each iteration of a test starts in the same system state.
- No other functions or applications are active in the system unless the scenario includes some amount of other activity going on in the system.

  Note: Started applications use memory even when they are minimized or idle. This increases the probability that paging will skew the results of the benchmark and violates the repeatability rule.
- The hardware and software used for benchmarking match your production environment.

For benchmarking, you create a scenario and then applications in this scenario several times, capturing key information during each run. Capturing key information after each run is of primary importance in determining the changes that might improve performance of both the application and the database.

Related concepts:

- "Benchmark preparation" on page 356
- "Benchmark test creation" on page 358
- "Benchmark test execution" on page 364
- "Benchmark test analysis example" on page 366

Benchmark preparation

Complete the logical design of the database against which the application runs before you start performance benchmarking. Set up and populate tables, views, and indexes. Normalize tables, bind application packages, and populate tables with realistic data.

You should also have determined the final physical design of the database. Place database manager objects in their final disk locations, size log files, determining the location of work files and backup, and test backup procedures. In addition, check packages to make sure that performance options such as row blocking are enabled when possible.

You should have reached a point in application programming and testing phases that will enable you to create your benchmark programs. Although the practical limits of an application might be revealed during the benchmark testing, 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, it should run 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.

Make sure that you run benchmark tests with a production-size database. An individual SQL statement should return as much data and require as much sorting as in production. This rule ensures that the application will test representative memory requirements.

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. Consider the frequency of execution and volume of data processed by the statements chosen average. If the volumes are excessive, consider the statements 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 tables 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.

Related concepts:

- "Benchmark testing" on page 355
- "Benchmark test creation" on page 358

Benchmark test creation

Consider a variety of factors when you design and implement a benchmark program. Because the main purpose of the program is to simulate a user application, the overall structure of the program varies. You might 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 might be more practical to include only blocks that contain the important statements.

To test the performance of specific SQL statements, you might 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 is simply to determine the time required to execute individual SQL statements.

For all benchmark testing, you need an efficient timing system to calculate the elapsed time, whether for individual SQL statements or the application as a whole. To simulate applications in which individual SQL statements are executed in isolation, it might be important to track times for CONNECT, PREPARE, and COMMIT statements. However, for programs that process many different statements, perhaps only a single CONNECT or COMMIT is necessary, and focusing on just the execution time for an individual statement might be the priority.

Although the elapsed time for each query is an important factor in performance analysis, it might not necessarily reveal bottlenecks. For example, information on CPU usage, locking, and buffer pool I/O might show that the application is I/O bound and is not 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 send the entire set of rows retrieved from a query to some output device. For example, the whole answer set might be input for another program, so that none of the rows from the first application are sent as output. Formatting data for screen output usually has high CPU cost and might not reflect user need. To provide an accurate simulation, a benchmark program should reflect the row handling of the specific application. If rows are 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 uses many of guidelines for creating a benchmark program. This tool can read SQL statements from either a flat file or standard input, dynamically describe and prepare the statements, and return an answer set. It also allows you to control the size of the answer set, as well as the number of rows that are sent from this answer set to an output device.

You can 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 also summarizes the performance results and provides both arithmetic and geometric means. For syntax and options, type db2batch -h on a command line.

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
```

It is possible to run db2batch remotely. If you use either the

```
-f <filename>
```

or the

```
-o <options>
```

command parameters of the benchmark tool then:

- The control options

  ```
  perf_detail
  ```

  and

  ```
  -p <perf_detail>
  ```

  (specifying the level of performance information to be returned) when set to greater than one are not supported when running remotely.

Other than these two items, the

```
perf_detail
```

and

```
-p <perf_detail>
```

control option values are supported and are valid for all DB2® Universal Database platforms.

Examples of db2batch tests

The following example shows 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 26. 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 27. 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 28. 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 29. 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 materialized query table is produced.

```
db2batch -d sample -f db2batch.sql -r /dev/null,
```

Produces just the materialized query table. Using the `-r` option, `outfile1` was replaced by `/dev/null` and `outfile2` (which contains just the materialized query 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 30. Sample Output from db2batch -- Materialized Query Table Only

Related concepts:

- "Benchmark test creation" on page 358
- "Benchmark test execution" on page 364
- "Benchmark test analysis example" on page 366

Benchmark test execution

For one type of database benchmark, you choose a configuration parameter and run 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, 20 or 30 times) with the same parameter value to get an average timing, which shows the effect of parameter changes more clearly.

When you run the benchmark, the first iteration, which is called a warm-up run, should be considered a separate case from the subsequent iterations, which are called normal runs. Because the warm-up run includes some start-up activities, such as initializing the buffer pool, and consequently, takes somewhat longer than normal runs. Although the information from the warm-up run might be realistically valid, it is not statistically valid. When you calculate the average timing or CPU for a specific set of parameter values, use only the results from normal runs.

You might consider using the Configuration Advisor to create the warm-up run of the benchmark. The questions that the Configuration Advisor asks can provide insight into some things to consider when you adjust the configuration of your environment for the normal runs during your benchmark activity. You can start the Configuration Advisor from the Control Center or by executing the db2 autoconfigure command with appropriate options.

If benchmarking uses individual queries, ensure that you minimize the potential effects of previous queries by flushing the buffer pool. To flush the buffer pool, read a number of pages that irrelevant to your query and to fill the buffer pool.

After you complete the iterations for a single set of parameter values, you can change a single parameter. However, between each iteration, perform the following tasks to restore the benchmark environment to its original state:

- . If the catalog statistics were updated for the test, make sure that the same values for the statistics are used for every iteration.
- The data used in the tests must be consistent if it is updated by the tests. This can be done by:
  - Using the RESTORE utility to restore the entire database. The backup copy of the database contains its previous state, 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 that contain this data.
- To return the application to its original state, re-bind it to the database.

In summary, follow these steps or iterations 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.
- 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.

Related concepts:

- "Benchmark testing" on page 355
- "Benchmark preparation" on page 356
- "Examples of db2batch tests" on page 360

Benchmark test analysis example

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 31. Benchmark Sample Results

Note: The data in the above report is shown for illustration purposes only. It does **not** represent measured results.

Analysis shows 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.

If your program can output data in a delimited ASCII format, 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
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 32. Benchmark Sample Timings Report

Note: The data in the above report is shown for illustration purposes only. It does **not** represent any measured results.

Related concepts:

- “Benchmark testing” on page 355
- “Benchmark test creation” on page 358
- “Benchmark test execution” on page 364

Chapter 13. Configuring DB2

Configuration parameters

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

Configuration files contain parameters that define values such as the resources allocated to the DB2 products and to individual databases, and the diagnostic level. There are two types of configuration files:

- the database manager configuration file for each DB2 instance
- the database configuration file for each individual database.

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

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

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

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

Most of the parameters either affect the amount of system resources that will be allocated to a single instance of the database manager, or they configure the setup of the database manager and the different communications

subsystems based on environmental considerations. In addition, there are other parameters that serve informative purposes only and cannot be changed. All of these parameters have global applicability independent of any single database stored under that instance of the database manager.

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

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

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

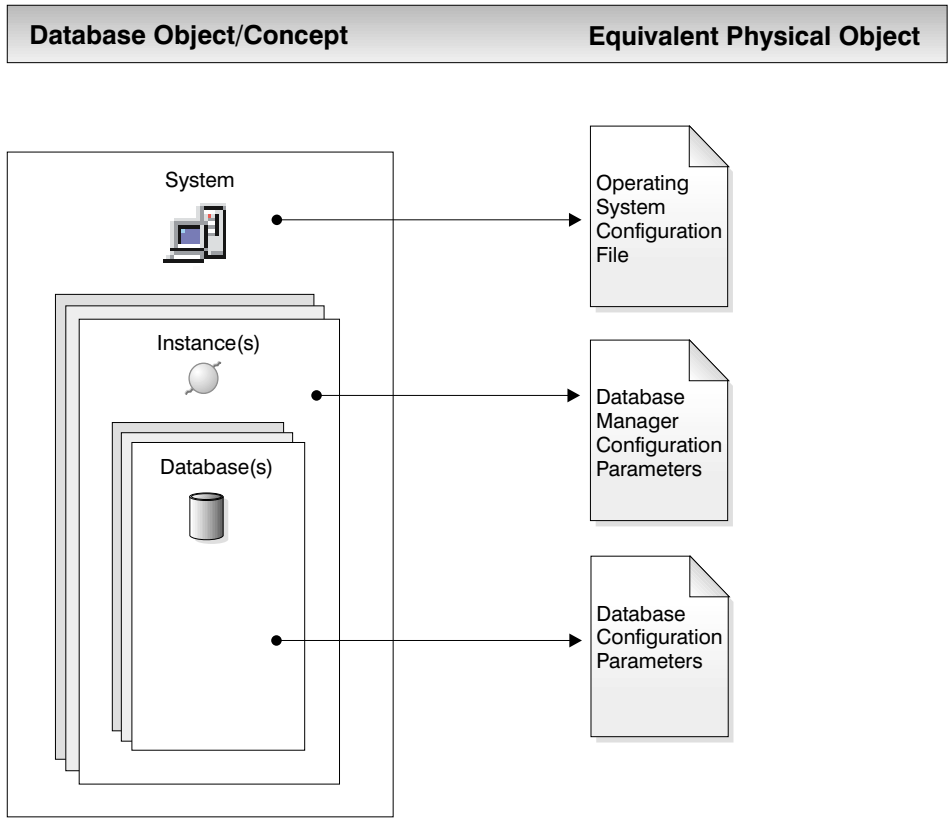


Figure 33. Configuration Parameter Files

Related concepts:

- "Configuration parameter tuning" on page 371

Related tasks:

- "Configuring DB2 with configuration parameters" on page 372

Configuration parameter tuning

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.

A good starting point for tuning your configuration is the Performance Configuration wizard or the AUTOCONFIGURE command.

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.

Some configuration parameters can be set to *automatic*. DB2® will then automatically adjust these parameters to reflect the current resource requirements.

Related concepts:

- "Configuration parameters" on page 369

Related tasks:

- "Configuring DB2 with configuration parameters" on page 372

Related reference:

- "Configuration parameters summary" on page 376

Configuring DB2 with configuration parameters

Database manager configuration parameters are stored in a file named `db2systm`. Database configuration parameters are stored in a file named `SQLDBCON`. These files 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 `db2systm` or `SQLDBCON` using a method other than those provided by DB2, you may make the database unusable. **We strongly**

recommend that you do not change these files using methods other than those documented and supported by DB2.

You may use one of the following methods to reset, update, and view configuration parameters:

- Using the Control Center. The Configure Instance notebook can be used to set the database manager configuration parameters on either a client or a server. The Configure Database notebook can be used to alter the value of database configuration parameters. The DB2 Control Center also provides the Configuration Advisor to alter the value of configuration parameters. This advisor generates values for 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.

  In a partitioned database environment, the SQLDBCON file exists for each database partition. The 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. (We recommend, however, that the configuration parameter values be the same on all partitions.)

  Note: The Configuration Advisor is not available in the partitioned database environment.

- Using the command line processor. Commands to change the settings can be quickly and conveniently entered:

  For database manager configuration parameters:
  - GET DATABASE MANAGER CONFIGURATION (or GET DBM CFG)
  - UPDATE DATABASE MANAGER CONFIGURATION (or UPDATE DBM CFG)
  - RESET DATABASE MANAGER CONFIGURATION (or RESET DBM CFG) to reset *all* database manager parameters to their default values
  - AUTOCONFIGURE.

  For database configuration parameters:
  - GET DATABASE CONFIGURATION (or GET DB CFG)
  - UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (or RESET DB CFG) to reset *all* database parameters to their default values
  - AUTOCONFIGURE.

- Using the application programming interfaces (APIs). The APIs can easily be called from an application or a host-language program.

- Using the Configuration Assistant (for database manager configuration parameters). You can only use the Configuration Assistant to set the database manager configuration parameters on a client.

For some database manager configuration parameters, the database manager must be stopped (db2stop) and then restarted (db2start) for the new parameter values to take effect.

For some database parameters, changes will only take effect when the database is reactivated. In these cases, 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.

Other parameters can be changed online; these are called *configurable online configuration parameters*.

If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the UPDATE DBM CFG command will be to apply the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

To change a database manager configuration parameter online:

```
db2 attach to <instance-name>
db2 update dbm cfg using <parameter-name> <value>
db2 detach
```

For clients, changes to the database manager configuration parameters take effect the next time the client connects to a server.

If you change a configurable online database configuration parameter while connected, the default behavior is to apply the change online, wherever possible. You should note that some parameter changes may take a noticeable amount of time to take effect due to the overhead associated with allocating space. To change configuration parameters online from the command line processor, a connection to the database is required. To change a database configuration parameter online:

```
db2 connect to <dbname>
db2 update db cfg using <parameter-name> <parameter-value>
db2 connect reset
```

Each configurable online configuration parameter has a *propagation class* associated with it. The propagation class indicates when you can expect a change to the configuration parameter to take effect. There are three propagation classes:

- **Immediate**: Parameters that change immediately upon command or API invocation. For example, *diaglevel* has a propagation class of immediate.
- **Statement boundary**: Parameters that change on statement and statement-like boundaries. For example, if you change the value of *sortheap*, all new SQL requests will start using the new value.
- **Transaction boundary**: Parameters that change on transaction boundaries. For example, a new value for *dl_expint* is updated after a COMMIT statement.

Changing some database configuration parameters can influence the access plan chosen by the SQL optimizer. After changing any of these parameters, you should consider rebinding your applications to ensure the best access plan is being used for your SQL statements. Any parameters that were modified online (for example, by using the UPDATE DATABASE CONFIGURATION IMMEDIATE command) will cause the SQL optimizer to choose new access plans for new SQL statements. However, the SQL statement cache will not be purged of existing entries. To clear the contents of the SQL cache, use the FLUSH PACKAGE CACHE statement.

While new parameter values may not be immediately effective, viewing the parameter settings (using GET DATABASE MANAGER CONFIGURATION or GET DATABASE CONFIGURATION commands) will always show the latest updates. Viewing the parameter settings using the SHOW DETAIL clause on these commands will show both the latest updates and the values in memory.

Note: A number of 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 documentation. To clarify what may be confusing, “Yes” should be considered equivalent to “On” and “No” should be considered equivalent to “Off”.

Related concepts:

- “Configuration parameters” on page 369
- “Configuration parameter tuning” on page 371

Related reference:

- “GET DATABASE CONFIGURATION Command” in the *Command Reference*
- “GET DATABASE MANAGER CONFIGURATION Command” in the *Command Reference*
- “RESET DATABASE CONFIGURATION Command” in the *Command Reference*
- “RESET DATABASE MANAGER CONFIGURATION Command” in the *Command Reference*

- "UPDATE DATABASE CONFIGURATION Command" in the *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION Command" in the *Command Reference*
- "Configuration parameters summary" on page 376
- "FLUSH PACKAGE CACHE statement" in the *SQL Reference, Volume 2*

Configuration parameters summary

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 "Automatic" in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DBM CFG command. If you set a parameter to automatic, DB2 will automtically adjust the parameter to reflect current resource requirements.

The column "Performance Impact" 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 25. Configurable Database Manager Configuration Parameters

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
agentpri	No	No	High	"Priority of Agents configuration parameter - agentpri" on page 443

Table 25. Configurable Database Manager Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
agent_stack_sz	No	No	Low	"Agent Stack Size configuration parameter - agent_stack_sz" on page 412
aslheapsz	No	No	High	"Application Support Layer Heap Size configuration parameter - aslheapsz" on page 417
audit_buf_sz	No	No	High	"Audit Buffer Size configuration parameter - audit_buf_sz" on page 425
authentication	No	No	Low	"Authentication Type configuration parameter - authentication" on page 523
catalog_noauth	Yes	No	None	"Cataloging Allowed without Authority configuration parameter - catalog_noauth" on page 526
comm_bandwidth	Yes	No	Medium	"Communications Bandwidth configuration parameter - comm_bandwidth" on page 513
conn_elapse	Yes	No	Medium	"Connection Elapse Time configuration parameter - conn_elapse" on page 501
cpuspeed	Yes	No	Low (see note)	"CPU Speed configuration parameter - cpuspeed" on page 513
datalinks	No	No	Low	"Enable Data Links Support configuration parameter - datalinks" on page 486
dft_account_str	Yes	No	None	"Default Charge-Back Account configuration parameter - dft_account_str" on page 518
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_timestamp* • *dft_mon_uow*	Yes	No	Medium	"Default Database System Monitor Switches configuration parameter - dft_monswitches" on page 511
dftdbpath	Yes	No	None	"Default Database Path configuration parameter - dftdbpath" on page 526

Table 25. Configurable Database Manager Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
diaglevel	Yes	No	Low	"Diagnostic Error Capture Level configuration parameter - diaglevel" on page 507
diagpath	Yes	No	None	"Diagnostic Data Directory Path configuration parameter - diagpath" on page 508
dir_cache	No	No	Medium	"Directory Cache Support configuration parameter - dir_cache" on page 423
discover	No	No	Medium	"Discovery Mode configuration parameter - discover" on page 499
discover_comm	No	No	Low	"Search Discovery Communications Protocols configuration parameter - discover_comm" on page 499
discover_inst	Yes	No	Low	"Discover Server Instance configuration parameter - discover_inst" on page 500
fcm_num_buffers	Yes	No	Medium	"Number of FCM Buffers configuration parameter - fcm_num_buffers" on page 502
fed_noauth	Yes	No	None	"Bypass Federated Authentication configuration parameter - fed_noauth" on page 525
federated	No	No	Medium	"Federated Database System Support configuration parameter - federated" on page 520
fenced_pool	No	No	Medium	"Maximum Number of Fenced Processes configuration parameter - fenced_pool" on page 452
health_mon	Yes	No	Low	"Health Monitoring configuration parameter - health_mon" on page 510
indexrec	Yes	No	Medium	"Index Re-creation Time configuration parameter - indexrec" on page 470
instance_memory	No	Yes	Medium	"Instance Memory configuration parameter - instance_memory" on page 421
intra_parallel	No	No	High	"Enable Intra-Partition Parallelism configuration parameter - intra_parallel" on page 506

Table 25. Configurable Database Manager Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
java_heap_sz	No	No	High	"Maximum Java Interpreter Heap Size configuration parameter - java_heap_sz" on page 426
jdk_path	No	No	None	"Java Development Kit Installation Path configuration parameter - jdk_path" on page 519
keepfenced	No	No	Medium	"Keep Fenced Process configuration parameter - keepfenced" on page 450
maxagents	No	No	Medium	"Maximum Number of Agents configuration parameter - maxagents" on page 444
maxcagents	No	No	Medium	"Maximum Number of Concurrent Agents configuration parameter - maxcagents" on page 445
max_connections	No	No	Medium	"Maximum Number of Client Connections configuration parameter - max_connections" on page 448
max_connretries	Yes	No	Medium	"Node Connection Retries configuration parameter - max_connretries" on page 503
max_coordagents	No	No	Medium	"Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
max_querydegree	Yes	No	High	"Maximum Query Degree of Parallelism configuration parameter - max_querydegree" on page 505
max_time_diff	No	No	Medium	"Maximum Time Difference Among Nodes configuration parameter - max_time_diff" on page 504
maxtotfilop	No	No	Medium	"Maximum Total Files Open configuration parameter - maxtotfilop" on page 442
min_priv_mem	No	No	Medium	"Minimum Committed Private Memory configuration parameter - min_priv_mem" on page 414
mon_heap_sz	No	No	Low	"Database System Monitor Heap Size configuration parameter - mon_heap_sz" on page 422

Table 25. Configurable Database Manager Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
nname	No	No	None	"NetBIOS Workstation Name configuration parameter - nname" on page 496
notifylevel	Yes	No	Low	"Notify Level configuration parameter - notifylevel" on page 509
numdb	No	No	Low	"Maximum Number of Concurrently Active Databases configuration parameter - numdb" on page 514
num_initagents	No	No	Medium	"Initial Number of Agents in Pool configuration parameter - num_initagents" on page 450
num_initfenced	No	No	Medium	"Initial Number of Fenced Processes configuration parameter - num_initfenced" on page 453
num_poolagents	No	No	High	"Agent Pool Size configuration parameter - num_poolagents" on page 448
priv_mem_thresh	No	No	Medium	"Private Memory Threshold configuration parameter - priv_mem_thresh" on page 415
query_heap_sz	No	No	Medium	"Query Heap Size configuration parameter - query_heap_sz" on page 411
resync_interval	No	No	None	"Transaction Resync Interval configuration parameter - resync_interval" on page 477
rqrioblk	No	No	High	"Client I/O Block Size configuration parameter - rqrioblk" on page 420
sheapthres	No	No	High	"Sort Heap Threshold configuration parameter - sheapthres" on page 406
spm_log_file_sz	No	No	Low	"Sync Point Manager Log File Size configuration parameter - spm_log_file_sz" on page 479
spm_log_path	No	No	Medium	"Sync Point Manager Log File Path configuration parameter - spm_log_path" on page 478
spm_max_resync	No	No	Low	"Sync Point Manager Resync Agent Limit configuration parameter - spm_max_resync" on page 479

Table 25. Configurable Database Manager Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
spm_name	No	No	None	"Sync Point Manager Name configuration parameter - spm_name" on page 478
start_stop_time	Yes	No	Low	"Start and Stop Timeout configuration parameter - start_stop_time" on page 504
svcename	No	No	None	"TCP/IP Service Name configuration parameter - svcename" on page 497
sysadm_group	No	No	None	"System Administration Authority Group Name configuration parameter - sysadm_group" on page 521
sysctrl_group	No	No	None	"System Control Authority Group Name configuration parameter - sysctrl_group" on page 522
sysmaint_group	No	No	None	"System Maintenance Authority Group Name configuration parameter - sysmaint_group" on page 523
tm_database	No	No	None	"Transaction Manager Database Name configuration parameter - tm_database" on page 476
tp_mon_name	No	No	None	"Transaction Processor Monitor Name configuration parameter - tp_mon_name" on page 516
tpname	No	No	None	"APPC Transaction Program Name configuration parameter - tpname" on page 498
trust_allclnts	No	No	None	"Trust All Clients configuration parameter - trust_allclnts" on page 527
trust_clntauth	No	No	None	"Trusted Clients Authentication configuration parameter - trust_clntauth" on page 528
use_sna_auth	Yes	No	None	"Use SNA Authentication configuration parameter - use_sna_auth" on page 525

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 26. Informational Database Manager Configuration Parameters

Parameter	Additional Information
nodetype	"Machine Node Type configuration parameter - nodetype" on page 518
release	"Configuration File Release Level configuration parameter - release" on page 481

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 "Automatic" in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DBM CFG command. If you set a parameter to automatic, DB2 will automtically adjust the parameter to reflect current resource requirements.

The column "Performance Impact" 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 27. Configurable Database Configuration Parameters

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
app_ctl_heap_sz	No	No	Medium	"Application Control Heap Size configuration parameter - app_ctl_heap_sz" on page 403
appgroup_mem_sz	No	No	Medium	"Maximum Size of Application Group Memory Set configuration parameter - appgroup_mem_sz" on page 402

Table 27. Configurable Database Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
applheapsz	No	No	Medium	"Application Heap Size configuration parameter - applheapsz" on page 410
autorestart	Yes	No	Low	"Auto Restart Enable configuration parameter - autorestart" on page 470
avg_appls	Yes	No	High	"Average Number of Active Applications configuration parameter - avg_appls" on page 440
blk_log_dsk_ful	Yes	No	None	"Block on Log Disk Full configuration parameter - blk_log_dsk_ful" on page 469
catalogcache_sz	Yes	No	High	"Catalog Cache Size configuration parameter - catalogcache_sz" on page 393
chngpgs_thresh	No	No	High	"Changed Pages Threshold configuration parameter - chngpgs_thresh" on page 431
database_memory	No	Yes	Medium	"Database Shared Memory Size configuration parameter - database_memory" on page 391
dbheap	Yes	No	Medium	"Database Heap configuration parameter - dbheap" on page 392
dft_degree	Yes	No	High	"Default Degree configuration parameter - dft_degree" on page 491
dft_extent_sz	Yes	No	Medium	"Default Extent Size of Table Spaces configuration parameter - dft_extent_sz" on page 436
dft_loadrec_ses	Yes	No	Medium	"Default Number of Load Recovery Sessions configuration parameter - dft_loadrec_ses" on page 472
dft_prefetch_sz	Yes	No	Medium	"Default Prefetch Size configuration parameter - dft_prefetch_sz" on page 435
dft_queryopt	Yes	No	Medium	"Default Query Optimization Class configuration parameter - dft_queryopt" on page 491
dft_refresh_age	No	No	Medium	"Default Refresh Age configuration parameter - dft_refresh_age" on page 492

Table 27. Configurable Database Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
dft_sqlmathwarn	No	No	None	"Continue upon Arithmetic Exceptions configuration parameter - dft_sqlmathwarn" on page 489
discover_db	Yes	No	Medium	"Discover Database configuration parameter - discover_db" on page 498
dlchktime	Yes	No	Medium	"Time Interval for Checking Deadlock configuration parameter - dlchktime" on page 427
dl_expint	Yes	No	None	"Data Links Access Token Expiry Interval configuration parameter - dl_expint" on page 484
dl_num_copies	Yes	No	None	"Data Links Number of Copies configuration parameter - dl_num_copies" on page 485
dl_time_drop	Yes	No	None	"Data Links Time After Drop configuration parameter - dl_time_drop" on page 485
dl_token	Yes	No	Low	"Data Links Token Algorithm configuration parameter - dl_token" on page 485
dl_upper	Yes	No	None	"Data Links Token in Upper Case configuration parameter - dl_upper" on page 486
dl_wt_iexpint	Yes	No	None	"Data Links Write Token Initial Expiry Interval configuration parameter - dl_wt_iexpint" on page 484
dyn_query_mgmt	No	No	Low	"Dynamic SQL Query Management configuration parameter - dyn_query_mgmt" on page 480
estore_seg_sz	No	No	Medium	"Extended Storage Memory Segment Size configuration parameter - estore_seg_sz" on page 437
groupheap_ratio	No	No	Medium	"Percent of Memory for Application Group Heap configuration parameter - groupheap_ratio" on page 403
indexrec	Yes	No	Medium	"Index Re-creation Time configuration parameter - indexrec" on page 470
locklist	Yes	No	High when it affects escalation	"Maximum Storage for Lock List configuration parameter - locklist" on page 397

Table 27. Configurable Database Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
locktimeout	No	No	Medium	"Lock Timeout configuration parameter - locktimeout" on page 430
logbufsz	No	No	High	"Log Buffer Size configuration parameter - logbufsz" on page 395
logfilsiz	No	No	Medium	"Size of Log Files configuration parameter - logfilsiz" on page 454
logprimary	No	No	Medium	"Number of Primary Log Files configuration parameter - logprimary" on page 456
logretain	No	No	Low	"Log Retain Enable configuration parameter - logretain" on page 467
logsecond	Yes	No	Medium	"Number of Secondary Log Files configuration parameter - logsecond" on page 458
maxappls	Yes	Yes	Medium	"Maximum Number of Active Applications configuration parameter - maxappls" on page 438
maxfilop	Yes	No	Medium	"Maximum Database Files Open per Application configuration parameter - maxfilop" on page 441
maxlocks	Yes	No	High when it affects escalation	"Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
mincommit	Yes	No	High	"Number of Commits to Group configuration parameter - mincommit" on page 464
min_dec_div_3	No	No	High	"Decimal Division Scale to 3 configuration parameter - min_dec_div_3" on page 419
newlogpath	No	No	Low	"Change the Database Log Path configuration parameter - newlogpath" on page 459
mirrorlogpath	No	No	Low	"Mirror Log Path configuration parameter - mirrorlogpath" on page 461
num_db_backups	Yes	No	None	"Number of Database Backups configuration parameter - num_db_backups" on page 473

Table 27. Configurable Database Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
num_estore_segs	No	No	Medium	"Number of Extended Storage Memory Segments configuration parameter - num_estore_segs" on page 437
num_freqvalues	Yes	No	Low	"Number of Frequent Values Retained configuration parameter - num_freqvalues" on page 493
num_iocleaners	No	No	High	"Number of Asynchronous Page Cleaners configuration parameter - num_iocleaners" on page 432
num_ioservers	No	No	High	"Number of I/O Servers configuration parameter - num_ioservers" on page 434
num_quantiles	Yes	No	Low	"Number of Quantiles for Columns configuration parameter - num_quantiles" on page 494
overflowlogpath	No	No	Medium	"Overflow Log Path configuration parameter - overflowlogpath" on page 462
pckcachesz	Yes	No	High	"Package Cache Size configuration parameter - pckcachesz" on page 399
rec_his_retentn	No	No	None	"Recovery History Retention Period configuration parameter - rec_his_retentn" on page 473
seqdetect	Yes	No	High	"Sequential Detection Flag configuration parameter - seqdetect" on page 434
sheapthres_shr	No	No	High	"Sort Heap Threshold for Shared Sorts configuration parameter - sheapthres_shr" on page 408
softmax	No	No	Medium	"Recovery Range and Soft Checkpoint Interval configuration parameter - softmax" on page 465
sortheap	Yes	No	High	"Sort Heap Size configuration parameter - sortheap" on page 405
stat_heap_sz	No	No	Low	"Statistics Heap Size configuration parameter - stat_heap_sz" on page 410
stmtheap	Yes	No	Medium	"Statement Heap Size configuration parameter - stmtheap" on page 409

Table 27. Configurable Database Configuration Parameters (continued)

Parameter	Configurable Online	Automatic	Performance Impact	Additional Information
trackmod	No	No	Low	"Track Modified Pages Enable configuration parameter - trackmod" on page 474
tsm_mgmtclass	Yes	No	None	"Tivoli Storage Manager Management Class configuration parameter - tsm_mgmtclass" on page 474
tsm_nodename	Yes	No	None	"Tivoli Storage Manager Node Name configuration parameter - tsm_nodename" on page 475
tsm_owner	Yes	No	None	"Tivoli Storage Manager Owner Name configuration parameter - tsm_owner" on page 476
tsm_password	Yes	No	None	"Tivoli Storage Manager Password configuration parameter - tsm_password" on page 475
userexit	No	No	Low	"User Exit Enable configuration parameter - userexit" on page 468
util_heap_sz	Yes	No	Low	"Utility Heap Size configuration parameter - util_heap_sz" on page 396
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 28. Informational Database Configuration Parameters

Parameter	Additional Information
backup_pending	"Backup Pending Indicator configuration parameter - backup_pending" on page 487
codepage	"Code Page for the Database configuration parameter - codepage" on page 482
codeset	"Codeset for the Database configuration parameter - codeset" on page 482
collate_info	"Collating Information configuration parameter - collate_info" on page 483
country	"Database Territory Code configuration parameter" on page 482
database_consistent	"Database is Consistent configuration parameter - database_consistent" on page 487
database_level	"Database Release Level configuration parameter - database_level" on page 481

Table 28. Informational Database Configuration Parameters (continued)

Parameter	Additional Information
log_retain_status	"Log Retain Status Indicator configuration parameter - log_retain_status" on page 488
loghead	"First Active Log File configuration parameter - loghead" on page 464
logpath	"Location of Log Files configuration parameter - logpath" on page 463
multipage_alloc	"Multipage File Allocation Enabled configuration parameter - multipage_alloc" on page 488
numsegs	"Default Number of SMS Containers configuration parameter - numsegs" on page 436
release	"Configuration File Release Level configuration parameter - release" on page 481
restore_pending	"Restore Pending configuration parameter - restore_pending" on page 488
rollfwd_pending	"Roll Forward Pending Indicator configuration parameter - rollfwd_pending" on page 487
territory	"Database Territory parameter - territory" on page 482
user_exit_status	"User Exit Status Indicator configuration parameter - user_exit_status" on page 488

DB2 Administration Server (DAS) Configuration Parameter Summary

Table 29. DAS Configuration Parameters

Parameter	Parameter Type	Additional Information
authentication	Configurable	"Authentication Type DAS configuration parameter - authentication" on page 537
contact_host	Configurable Online	"Location of Contact List configuration parameter - contact_host" on page 536
das_codepage	Configurable Online	"DAS Code Page configuration parameter - das_codepage" on page 537
das_territory	Configurable Online	"DAS Territory configuration parameter - das_territory" on page 538

Table 29. DAS Configuration Parameters (continued)

Parameter	Parameter Type	Additional Information
dasadm_group	Configurable	"DAS Administration Authority Group Name configuration parameter - dasadm_group" on page 531
db2system	Configurable Online	"Name of the DB2 Server System configuration parameter - db2system" on page 530
discover	Configurable Online	"DAS Discovery Mode configuration parameter - discover" on page 530
exec_exp_task	Configurable	"Execute Expired Tasks configuration parameter - exec_exp_task" on page 535
jdk_path	Configurable Online	"Java Development Kit Installation Path DAS configuration parameter - jdk_path" on page 535
sched_enable	Configurable	"Scheduler Mode configuration parameter - sched_enable" on page 532
sched_userid	Informational	"Scheduler User ID configuration parameter - sched_userid" on page 536
smtp_server	Configurable Online	"SMTP Server configuration parameter - smtp_server" on page 534
toolscat_db	Configurable	"Tools Catalog Database configuration parameter - toolscat_db" on page 533
toolscat_inst	Configurable	"Tools Catalog Database Instance configuration parameter - toolscat_inst" on page 532
toolscat_schema	Configurable	"Tools Catalog Database Schema configuration parameter - toolscat_schema" on page 533

Parameter Details by Function

The 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 391
- "Logging and Recovery" on page 454
- "Database Management" on page 480
- "Communications" on page 496
- "Partitioned Database Environment" on page 501
- "Instance Management" on page 507
- "DB2 Administration Server" on page 529

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, and whether the change will take effect online:

- *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.

- *Configurable online*

  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. Changes can be applied while the database is online, without having to stop and restart the database manager, or reactivate the database.

- *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 401
- "Agent Private Memory" on page 405
- "Agent/Application Communication Memory" on page 416
- "Database Manager Instance Memory" on page 421
- "Locks" on page 427
- "I/O and Storage" on page 431
- "Agents" on page 438
- "Stored Procedures and User Defined Functions" on page 450.

Database Shared Memory

The following parameters affect the database global memory allocated on your system:

- "Database Shared Memory Size configuration parameter - database_memory".
- "Database Heap configuration parameter - dbheap" on page 392.
- "Catalog Cache Size configuration parameter - catalogcache_sz" on page 393.
- "Log Buffer Size configuration parameter - logbufsz" on page 395.
- "Utility Heap Size configuration parameter - util_heap_sz" on page 396.
- "Maximum Storage for Lock List configuration parameter - locklist" on page 397.
- "Package Cache Size configuration parameter - pckcachesz" on page 399.

Database Shared Memory Size configuration parameter - database_memory

Configuration Type Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type	Configurable
Default [Range]	Automatic [0 — 4 294 967 295]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is activated
When Freed	When the database is deactivated

This parameter controls the minimum amount of shared memory that is reserved for a given database. It allows the database administrator to specify a minimum amount of shared memory for each database. The database memory does not include the database manager shared memory, or the application group memory.

To simplify the management of this parameter, the AUTOMATIC setting instructs DB2 to calculate the amount of memory needed, and to allocate it at database activation time. On 64-bit AIX, the AUTOMATIC value also permits DB2 to grow its memory usage as needed, when the buffer pools grow, or when additional memory is needed for control blocks.

Recommendation: This value will usually remain at AUTOMATIC. However, it can be used to reserve additional memory for future expansion. For example, the additional memory can be used for creating new buffer pools, or increasing the sizes of existing buffer pools.

Related reference:

- “Package Cache Size configuration parameter - pckcachesz” on page 399
- “Maximum Storage for Lock List configuration parameter - locklist” on page 397
- “Database Heap configuration parameter - dbheap” on page 392
- “Utility Heap Size configuration parameter - util_heap_sz” on page 396
- “Catalog Cache Size configuration parameter - catalogcache_sz” on page 393
- “Instance Memory configuration parameter - instance_memory” on page 421
- “Sort Heap Threshold for Shared Sorts configuration parameter - sheapthres_shr” on page 408

Database Heap configuration parameter - dbheap

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate

Default [Range]	Automatic [32 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is activated
When Freed	When the database is deactivated

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 event monitor buffers, the log buffer (*logbufsz*), and temporary memory used by utilities. Therefore, the size of the heap will be dependent on a large number of variables. 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.

You can use the database system monitor to track the highest amount of memory that was used for the database heap, using the db_heap_top (maximum database heap allocated) element.

Related reference:

- "Log Buffer Size configuration parameter - logbufsz" on page 395
- "Maximum Database Heap Allocated" in the *System Monitor Guide and Reference*

Catalog Cache Size configuration parameter - catalogcache_sz

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	-1 [8 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is initialized
When Freed	When the database is shut down

This parameter is allocated out of the database shared memory, and is used to cache system catalog information. In a partitioned database system, there is one catalog cache for each database partition.

Caching catalog information at individual partitions allows the database manager to reduce its internal overhead by eliminating the need to access the

system catalogs (and/or the catalog node in an partitioned database environment) to obtain information that has previously been retrieved. The catalog cache is used to store:

- SYSTABLES information (including packed descriptors)
- authorization information, including SYSDBAUTH information and execute privileges for routines
- SYSROUTINES information

The use of the catalog cache can help improve the overall performance of:

- binding packages and compiling SQL statements
- operations that involve checking database-level privileges
- operations that involve checking execute privileges for routines
- applications that are connected to non-catalog nodes in a partitioned database environment

By taking the default (-1) in a server or partitioned database environment, the value used to calculate the page allocation is four times the value specified for the *maxappls* configuration parameter. The exception to this occurs if four times *maxappls* is less than 8. In this situation, the default value of -1 will set *catalogcache_sz* to 8.

Recommendation: Start with the default value and tune it by using the database system monitor. When tuning this parameter, you should consider whether the extra memory being reserved for the catalog cache might be more effective if it was allocated for another purpose, such as the buffer pool or package cache.

Tuning this parameter is particularly important if a workload involves many SQL compilations for a brief period of time, with few or no SQL compilations thereafter. If the cache is too large, memory may be wasted holding copies of information that will no longer be used.

In an partitioned database environment, consider if the *catalogcache_sz* at the catalog node needs to be set larger since catalog information that is required at non-catalog nodes will always first be cached at the catalog node.

The *cat_cache_lookups* (catalog cache lookups), *cat_cache_inserts* (catalog cache inserts), *cat_cache_overflows* (catalog cache overflows), and *cat_cache_size_top* (catalog cache high water mark) monitor elements can help you determine whether you should adjust this configuration parameter.

Note: The catalog cache exists on all nodes in a partitioned database environment. Since there is a local database configuration file for each node, each node's *catalogcache_sz* value defines the size of the local

catalog cache. In order to provide efficient caching and avoid overflow scenarios, you need to explicitly set the *catalogcache_sz* value at each node and consider the feasibility of possibly setting the *catalogcache_sz* on non-catalog nodes to be smaller than that of the catalog node; keep in mind that information that is required to be cached at non-catalog nodes will be retrieved from the catalog node's cache. Hence, a catalog cache at a non-catalog node is like a subset of the information in the catalog cache at the catalog node.

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 large number of static SQL statements.

Related reference:

- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438
- "Catalog Cache Lookups" in the *System Monitor Guide and Reference*
- "Catalog Cache Inserts" in the *System Monitor Guide and Reference*
- "Catalog Cache Overflows" in the *System Monitor Guide and Reference*
- "Catalog Cache High Water Mark" in the *System Monitor Guide and Reference*

Log Buffer Size configuration parameter - logbufsz

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	
	32-bit platforms 8 [4 — 4 096]
	64-bit platforms 8 [4 — 65 535]
Unit of Measure	Pages (4 KB)

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). Refer to the *log_space_used* (unit of work log space used) monitor element.

Related reference:

- "Number of Commits to Group configuration parameter - mincommit" on page 464
- "Database Heap configuration parameter - dbheap" on page 392
- "Unit of Work Log Space Used" in the *System Monitor Guide and Reference*

Utility Heap Size configuration parameter - util_heap_sz

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
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

This parameter indicates the maximum amount of memory that can be used simultaneously by the BACKUP, RESTORE, and LOAD (including 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.

Maximum Storage for Lock List configuration parameter - locklist

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	
	UNIX 100 [4 – 524 288]
	Windows Database server with local and remote clients 50 [4 – 524 288]
	Windows 64-bit Database server with local clients 50 [4 – 60 000]
	Windows 32-bit 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

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.

This parameter can be changed online, but it can only be increased online, not decreased. If you want to decrease the value of *locklist*, you will have to reactivate the database.

On 32-bit platforms, 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.

On 64-bit platforms, each lock requires 56 or 112 bytes of the lock list, depending on whether other locks are held on the object:
- 112 bytes are required to hold a lock on an object that has no other locks held on it

- 56 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 option of the ALTER TABLE statement to control how locking is done for a specific table.

  Use of the Repeatable Read isolation level may result in an automatic table lock.
- 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. Refer to the *lock_escals (lock escalations)* monitor element.

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, using *one* of the following calculations, depending on your environment:
   a. `(512 * x * maxappls) / 4096`
   b. with Concentrator enabled:

      `(512 * x * max_coordagents) / 4096`
   c. in a partitioned database with Concentrator enabled:

```
(512 * x * max_coordagents * number of database partitions) / 4096
```

where 512 is an estimate of the average number of locks per application and x is the number of bytes required for each lock against an object that has an existing lock (36 bytes on 32-bit platforms, 56 bytes on 64-bit platforms).

2. Calculate an upper bound for the size of your lock list:

   ```
   (512 * y * maxappls) / 4096
   ```

   where y is the number of bytes required for the first lock against an object (72 bytes on 32-bit platforms, 112 bytes on 64-bit platforms).
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. Refer to the *locks_held_top (maximum number of locks held)* monitor element.

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 command) after changing this parameter.

Related reference:

- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438
- "Number of Lock Escalations" in the *System Monitor Guide and Reference*
- "Maximum Number of Locks Held" in the *System Monitor Guide and Reference*
- "REBIND Command" in the *Command Reference*

Package Cache Size configuration parameter - pckcachesz

Configuration Type Database

Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	
	32-bit platforms -1 [-1, 32 — 128 000]
	64-bit platforms -1 [-1, 32 — 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is initialized
When Freed	When the database is shut down

This parameter is allocated out of the database shared memory, and is used for caching of sections for static and dynamic SQL statements on a database. In a partitioned database system, 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), 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 or catalog cache. 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.

The following monitor elements can help you determine whether you should adjust this configuration parameter:

- *pkg_cache_lookups* (package cache lookups)
- *pkg_cache_inserts* (package cache inserts)
- *pkg_cache_size_top* (package cache high water mark)
- *pkg_cache_num_overflows* (package cache overflows)

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.

Related reference:

- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438
- "Package Cache Lookups" in the *System Monitor Guide and Reference*
- "Package Cache Inserts" in the *System Monitor Guide and Reference*
- "Package Cache Overflows" in the *System Monitor Guide and Reference*
- "Package Cache High Water Mark" in the *System Monitor Guide and Reference*

Application Shared Memory

The following parameters specify the work area that is used by all agents (both coordinating and subagents) that work for an application:

- "Maximum Size of Application Group Memory Set configuration parameter - appgroup_mem_sz" on page 402
- "Percent of Memory for Application Group Heap configuration parameter - groupheap_ratio" on page 403
- "Application Control Heap Size configuration parameter - app_ctl_heap_sz" on page 403

Maximum Size of Application Group Memory Set configuration parameter - appgroup_mem_sz

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX Database server with local clients (other than 32-bit HP-UX)
20 000 [1 – 1 000 000]

32-bit HP-UX

- Database server with local clients
- Database server with local and remote clients
- Partitioned database server with local and remote clients

10 000 [1 – 1 000 000]

Windows Database server with local clients
10 000 [1 – 1 000 000]

Database server with local and remote clients (other than 32-bit HP-UX)
30 000 [1 – 1 000 000]

Partitioned database server with local and remote clients (other than 32-bit HP-UX)
40 000 [1 – 1 000 000]

Unit of Measure Pages (4 KB)

This parameter determines the size of the application group shared memory segment. Information that needs to be shared between agents working on the same application is stored in the application group shared memory segment.

In a partitioned database, or in a non-partitioned database with intra-partition parallelism enabled or concentrator enabled, multiple applications share one application group. One application group shared memory segment is allocated for the application group. Within the application group shared memory segment, each application will have its own application control heap, and all applications will share one application group shared heap.

The number of applications in one application group is calculated by:

```
appgroup_mem_sz / app_ctl_heap_sz
```

The application group shared heap size is calculated by:

```
appgroup_mem_sz * groupheap_ratio / 100
```

The size of each application control heap is calculated by:

```
app_ctl_heap_sz * (100 - groupheap_ratio) / 100
```

Recommendation: Retain the default value of this parameter unless you are experiencing performance problems.

Related reference:

- "Application Control Heap Size configuration parameter - app_ctl_heap_sz" on page 403
- "Percent of Memory for Application Group Heap configuration parameter - groupheap_ratio" on page 403

Percent of Memory for Application Group Heap configuration parameter - groupheap_ratio

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	70 [1 – 99]
Unit of Measure	Percentage

This parameter specifies the percentage of memory in the application control shared memory set devoted to the application group shared heap.

This parameter does not have any effect on a non-partitioned database with concentrator OFF and intra-partition parallelism disabled.

Recommendation: Retain the default value of this parameter unless you are experiencing performance problems.

Related reference:

- "Application Control Heap Size configuration parameter - app_ctl_heap_sz" on page 403
- "Maximum Size of Application Group Memory Set configuration parameter - appgroup_mem_sz" on page 402

Application Control Heap Size configuration parameter - 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

For partitioned databases, and for non-partitioned databases with intra-parallelism enabled (intra_parallel=ON), this parameter specifies the average size of the shared memory area allocated for an application. For non-partitioned databases where intra-parallelism is disabled (intra_parallel=OFF), this is the maximum private memory that will be allocated for the heap. There is one application control heap per connection per partition.

The application control heap is required primarily for sharing information between agents working on behalf of the same request. Usage of this heap is minimal for non-partitioned databases when running queries with a degree of parallelism equal to 1.

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.

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.

Related reference:

- “Enable Intra-Partition Parallelism configuration parameter - intra_parallel” on page 506
- “Application Heap Size configuration parameter - applheapsz” on page 410
- “Maximum Size of Application Group Memory Set configuration parameter - appgroup_mem_sz” on page 402
- “Percent of Memory for Application Group Heap configuration parameter - groupheap_ratio” on page 403

Agent Private Memory

The following parameters affect the amount of memory used for each database agent:

- “Sort Heap Size configuration parameter - sortheap”.
- “Sort Heap Threshold configuration parameter - sheapthres” on page 406.
- “Sort Heap Threshold for Shared Sorts configuration parameter - sheapthres_shr” on page 408.
- “Statement Heap Size configuration parameter - stmtheap” on page 409.
- “Application Heap Size configuration parameter - applheapsz” on page 410.
- “Statistics Heap Size configuration parameter - stat_heap_sz” on page 410.
- “Query Heap Size configuration parameter - query_heap_sz” on page 411.
- “Agent Stack Size configuration parameter - agent_stack_sz” on page 412.
- “Minimum Committed Private Memory configuration parameter - min_priv_mem” on page 414.
- “Private Memory Threshold configuration parameter - priv_mem_thresh” on page 415.
- “Maximum Java Interpreter Heap Size configuration parameter - java_heap_sz” on page 426.

Sort Heap Size configuration parameter - sortheap

Configuration Type Database

Parameter Type Configurable Online

Propagation Class Statement boundary

Default [Range]

32-bit platforms
256 [16 – 524 288]

64-bit platforms
256 [16 – 1 048 575]

Unit of Measure Pages (4 KB)

When Allocated As needed to perform sorts

When Freed When sorting is complete

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:

When working with the sort heap, you should consider the following:

- Appropriate indexes can minimize the use of the sort heap.
- Hash join buffers and dynamic bitmaps (used for index ANDing and Star Joins) use sort heap memory. Increase the size of this parameter when these techniques are used.
- 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 command) after changing this parameter.

Related reference:

- "Sort Heap Threshold configuration parameter - sheapthres" on page 406
- "REBIND Command" in the *Command Reference*
- "Sort Heap Threshold for Shared Sorts configuration parameter - sheapthres_shr" on page 408

Sort Heap Threshold configuration parameter - sheapthres

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX 32-bit platforms
20 000 [250 — 2 097 152]

Windows platforms
10 000 [250 — 2 097 152]

64-bit platforms
20 000 [250 — 2 147 483 647]

Unit of Measure Pages (4 KB)

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: sorts, hash joins, dynamic bitmaps (used for index ANDing and Star 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.

There is no reason to increase the value of this parameter when moving from a non-partitioned to a partitioned database environment. Once you have tuned the database and database manager configuration parameters on a single database partition environment, the same values will in most cases work well in a partitioned database environment.

The Sort Heap Threshold parameter, as a database manager configuration parameter, applies across the entire DB2 instance. The only way to set this parameter to different values on different nodes or partitions, is to create more than one DB2 instance. This will require managing different DB2 databases over different database partition groups. Such an arrangement defeats the purpose of many of the advantages of a partitioned database environment.

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.

You should use benchmarking techniques to tune this parameter to find the proper balance between sort performance and memory usage.

You can use the database system monitor to track the sort activity, using the post threshold sorts (*post_threshold_sorts*) monitor element.

Related reference:

- "Sort Heap Size configuration parameter - sortheap" on page 405
- "Post Threshold Sorts" in the *System Monitor Guide and Reference*
- "Sort Heap Threshold for Shared Sorts configuration parameter - sheapthres_shr" on page 408

Sort Heap Threshold for Shared Sorts configuration parameter - sheapthres_shr

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	
	32-bit platforms *sheapthres* [250 — 2 097 152]
	64-bit platforms *sheapthres* [250 — 2 147 483 647]
Unit of Measure	Pages (4 KB)

This parameter represents a hard limit on the total amount of database shared memory that can be used for sorting at any one time. When the total amount of shared memory for active shared sorts reaches this limit, subsequent sorts will fail (SQL0955C). If the value of *sheapthres_shr* is 0, the threshold for shared sort memory will be equal to the value of the *sheapthres* database

manager configuration parameter, which is also used to represent the sort memory threshold for private sorts. If the value of *sheapthres_shr* is non-zero, then this non-zero value will be used for the shared sort memory threshold.

sheapthres_shr is only meaningful in two cases:

- if the *intra_parallel* database manager configuration parameter is set to *yes*, because when *intra_parallel* is set to *no*, there will be no shared sorts.
- if the Concentrator is on (that is, when *max_connections* is greater than *max_coordagents*), because sorts that use a cursor declared with the WITH HOLD option will be allocated from shared memory.

Related reference:

- "Sort Heap Threshold configuration parameter - sheapthres" on page 406

Statement Heap Size configuration parameter - stmtheap

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Statement boundary
Default [Range]	2048 [128 – 65 535]
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.

Related reference:

- “Sort Heap Size configuration parameter - sortheap” on page 405
- “Application Heap Size configuration parameter - applheapsz” on page 410
- “Statistics Heap Size configuration parameter - stat_heap_sz” on page 410

Application Heap Size configuration parameter - applheapsz

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	256 [16 – 60 000]
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

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.

Related reference:

- “Application Control Heap Size configuration parameter - app_ctl_heap_sz” on page 403

Statistics Heap Size configuration parameter - 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

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.

Related reference:

- "Number of Frequent Values Retained configuration parameter - num_freqvalues" on page 493
- "Number of Quantiles for Columns configuration parameter - num_quantiles" on page 494
- "RUNSTATS Command" in the *Command Reference*

Query Heap Size configuration parameter - 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
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

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 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.

Related reference:

- "Application Support Layer Heap Size configuration parameter - aslheapsz" on page 417

Agent Stack Size configuration parameter - agent_stack_sz

Configuration Type	Database manager
Applies to	

	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	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 is used to set the initial committed stack size for each agent in a Windows environment. By default, each agent stack can grow up to the default reserve stack size of 256 KB (64 4-KB pages). This limit is sufficient for most database operations. However, when preparing a large SQL statement, the agent can run out of stack space and the system will generate a stack overflow exception (0xC000000D). When this happens, the server will shut down because the error is non-recoverable.

The agent stack size can be increased by setting *agent_stack_sz* to a value larger than the default reserve stack size of 64 pages. Note that the value for *agent_stack_sz*, when larger than the default reserve stack size, is rounded by the Windows operating system to the nearest multiple of 1 MB; setting the agent stack size to 128 4-KB pages actually reserves a 1 MB stack for each agent. Setting the value for *agent_stack_sz* less than the default reserve stack size will have no effect on the maximum limit because the stack still grows if necessary up to the default reserve stack size. In this case, the value for *agent_stack_sz* is the initial committed memory for the stack when an agent is created.

You can change the default reserve stack size by using the db2hdr utility to change the header information for the `db2syscs.exe` file. Changing the default reserve stack size will affect all threads while changing *agent_stack_sz* only affects the stack size for agents. The advantage of changing the default stack size using the db2hdr utility is that it provides a better granularity, therefore

allowing the stack size to be set at the minimum required stack size. However, you will have to stop and restart DB2 for a change to `db2syscs.exe` to take effect.

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.

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 Windows platforms.

This parameter does not apply to UNIX-based platforms.

Minimum Committed Private Memory configuration parameter - min_priv_mem

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	32 [32 – 112 000]
Unit of Measure	Pages (4 KB)
When Allocated	When the database manager is started
When Freed	When the database manager is stopped

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.

Related reference:

- "Private Memory Threshold configuration parameter - priv_mem_thresh" on page 415

Private Memory Threshold configuration parameter - priv_mem_thresh

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	1296 [-1; 32 – 112 000]
Unit of Measure	Pages (4 KB)

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.

Related reference:

- "Minimum Committed Private Memory configuration parameter - min_priv_mem" on page 414

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 configuration parameter - aslheapsz”
- “Decimal Division Scale to 3 configuration parameter - min_dec_div_3” on page 419
- “Client I/O Block Size configuration parameter - rqrioblk” on page 420

Application Support Layer Heap Size configuration parameter - aslheapsz

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	15 [1 – 524 288]
Unit of Measure	Pages (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

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 for two other purposes:

- It is 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.

- It is used to determine the communication size between agents and db2fmp processes. (A db2fmp process can be a user-defined function or a fenced stored procedure.) The number of bytes is allocated from shared memory for each db2fmp process or thread that is active on the system.

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 a minimum number of pages for *aslheapsz*:

```
aslheapsz >= ( sizeof(input SQLDA)
             + sizeof(each input SQLVAR)
             + sizeof(output SQLDA)
             + 250 ) / 4096
```

where sizeof(x) is the size of x in bytes that calculates the number of pages of a given input or output value.

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 4 096 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.

Related reference:

- "Query Heap Size configuration parameter - query_heap_sz" on page 411

Decimal Division Scale to 3 configuration parameter - min_dec_div_3

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Yes, No]

The *min_dec_div_3* database configuration parameter is provided as a quick way to enable a change to computation of the scale for decimal division in SQL. *min_dec_div_3* can be set to ″Yes″ or ″No″. The default value for *min_dec_div_3* is ″No″.

The *min_dec_div_3* database configuration parameter changes the resulting scale of a decimal arithmetic operation involving division. If the value is ″No″, the scale is calculated as 31-p+s-s'. If set to ″Yes″, the scale is calculated as MAX(3, 31-p+s-s'). This causes the result of decimal division to always have a scale of at least 3. Precision is always 31.

Changing this database configuration parameter may cause changes to applications for existing databases. This can occur when the resulting scale for decimal division would be impacted by changing this database configuration parameter. Listed below are some possible scenarios that may impact applications. These scenarios should be considered before changing the *min_dec_div_3* on a database server with existing databases.

- If the resulting scale of one of the view columns is changed, a view that is defined in an environment with one setting could fail with SQLCODE -344 when referenced after the database configuration parameter is changed. The message SQL0344N refers to recursive common table expressions, however, if the object name (first token) is a view, then you will need to drop the view and create it again to avoid this error.
- A static package will not change behavior until the package is rebound, either implicitly or explicitly. For example, after changing the value from NO to YES, the additional scale digits may not be included in the results until rebind occurs. For any changed static packages, an explicit REBIND command can be used to force a rebind.
- A check constraint involving decimal division may restrict some values that were previously accepted. Such rows now violate the constraint but will not be detected until one of the columns involved in the check constraint row is updated or the SET INTEGRITY statement with the IMMEDIATE CHECKED option is processed. To force checking of such a constraint, perform an ALTER TABLE statement in order to drop the check constraint and then perform an ALTER TABLE statement to add the constraint again.

Note: *min_dec_div_3* also has the following limitations:

1. The command `GET DB CFG FOR DBNAME` will not display the *min_dec_div_3* setting. The best way to determine the current setting

is to observe the side-effect of a decimal division result. For example, consider the following statement:

```
VALUES (DEC(1,31,0)/DEC(1,31,5))
```

If this statement returns sqlcode SQL0419N, then the database does not have *min_dec_div_3* support or it is set to ″No″. If the statement returns `1.000`, then *min_dec_div_3* is set to ″Yes″.

2. *min_dec_div_3* does not appear in the list of configuration keywords when you run the following command: `? UPDATE DB CFG`

Client I/O Block Size configuration parameter - rqrioblk

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 32 767 [4 096 – 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

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 32 767 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 32 767 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 4 096 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.

Database Manager Instance Memory

The following parameters affect memory that is allocated and used at an instance level:

- "Instance Memory configuration parameter - instance_memory"
- "Database System Monitor Heap Size configuration parameter - mon_heap_sz" on page 422
- "Directory Cache Support configuration parameter - dir_cache" on page 423
- "Audit Buffer Size configuration parameter - audit_buf_sz" on page 425
- "Maximum Java Interpreter Heap Size configuration parameter - java_heap_sz" on page 426

Instance Memory configuration parameter - instance_memory

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]	Automatic [8 — 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the instance is started
When Freed	When the instance is stopped

This parameter specifies the amount of memory which should be reserved for instance management. This includes memory areas that describe the databases on the instance.

If you set this parameter to *automatic*, DB2 will calculate the amount of instance memory needed for the current configuration.

Related reference:

- "Maximum Number of Agents configuration parameter - maxagents" on page 444
- "Maximum Number of Concurrently Active Databases configuration parameter - numdb" on page 514

Database System Monitor Heap Size configuration parameter - mon_heap_sz

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	**UNIX** 56 [0 – 60 000] **Windows Database server with local and remote clients** 32 [0 – 60 000] **Windows 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

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:

- When the first application connects to the database for which this event monitor is defined, an error message is written to the administration notification log.
- 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, an error code is returned to your application.

Related reference:

- "Default Database System Monitor Switches configuration parameter - dft_monswitches" on page 511

Directory Cache Support configuration parameter - dir_cache

Configuration Type Database manager

Applies to

- Database server with local and remote clients

- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type

Configurable

Default [Range]

Yes [Yes; No]

When Allocated

- When an application issues its first connect, the application directory cache is allocated
- When a database manager instance is started (`db2start`), the server directory cache is allocated.

When Freed

- When an the application process terminates, the application directory cache is freed
- When a database manager instance is stopped (`db2stop`), the server directory 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:

- An application directory cache that is allocated and used for each application process on the machine at which the application is running.
- A server directory cache that is allocated and used for some of the internal database manager processes.

For application directory 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 application directory 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 application directory 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 server directory caches, when a database manager instance is started (`db2start`), each directory file is read and the information is cached in the

server memory. This cache 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 server directory 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 configuration parameter - 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
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.

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 configuration parameter - 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
Parameter Type	Configurable
Default [Range]	512 [0 - 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When a Java stored procedure or UDF starts
When Freed	When the db2fmp process (fenced) or the db2agent process (trusted) terminates.

This parameter determines the maximum size of the heap that is used by the Java interpreter started to service Java DB2 stored procedures and UDFs.

There is one heap for each DB2 process (one for each agent or subagent on UNIX-based platforms, and one for each instance on other platforms). There is one heap for each fenced UDF and fenced stored procedure process. There is one heap per agent (not including sub-agents) for trusted routines. There is one heap per db2fmp process running a Java stored procedure. For multithreaded db2fmp processes, multiple applications using threadsafe fenced routines are serviced from a single heap. In all situations, only the

agents or processes that run Java UDFs or stored procedures ever allocate this memory. On partitioned database systems, the same value is used at each partition.

Related reference:

- "Java Development Kit Installation Path configuration parameter - jdk_path" on page 519

Locks

The following parameters influence how locking is managed in your environment:

- "Time Interval for Checking Deadlock configuration parameter - dlchktime"
- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
- "Lock Timeout configuration parameter - locktimeout" on page 430

See also "Maximum Storage for Lock List configuration parameter - locklist" on page 397.

Time Interval for Checking Deadlock configuration parameter - dlchktime

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	10 000 (10 seconds) [1 000 – 600 000]
Unit of Measure	Milliseconds

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 in more lock contention and as a result, more deadlock situations.

Related reference:

- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428

Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks

Configuration Type	Database	
Parameter Type	Configurable online	
Propagation Class	Immediate	
Default [Range]		
	UNIX	10 [1 – 100]
	Windows	22 [1 – 100]
Unit of Measure	Percentage	

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs escalation. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the *maxlocks* value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage

of the lock list held is below the value of *maxlocks*. The *maxlocks* parameter multiplied by the *maxappls* parameter cannot be less than 100.

Recommendation: The following formula allows you to set *maxlocks* to allow an application to hold twice the average number of locks:

```
maxlocks = 2 * 100 / maxappls
```

Where 2 is used to achieve twice the average and 100 represents the largest percentage value allowed. If you have only a few applications that run concurrently, you could use the following formula as an alternative to the first formula:

```
maxlocks = 2 * 100 / (average number of applications running
concurrently)
```

One of the considerations when setting *maxlocks* is to use it in conjunction with the size of the lock list (*locklist*). The actual limit of the number of locks held by an application before lock escalation occurs is:

```
maxlocks * locklist * 4 096 / (100 * 36) on a 32-bit system

maxlocks * locklist * 4 096 / (100 * 56) on a 64-bit system
```

Where 4 096 is the number of bytes in a page, 100 is the largest percentage value allowed for *maxlocks*, and 36 is the number of bytes per lock on a 32-bit system, and 56 is the number of bytes per lock on a 64-bit system. If you know that one of your applications requires 1 000 locks, and you do not want lock escalation to occur, then you should choose values for *maxlocks* and *locklist* in this formula so that the result is greater than 1 000. (Using 10 for *maxlocks* and 100 for *locklist*, this formula results in greater than the 1 000 locks needed.)

If *maxlocks* is set too low, lock escalation happens when there is still enough lock space for other concurrent applications. If *maxlocks* is set too high, a few applications can consume most of the lock space, and other applications will have to perform lock escalation. The need for lock escalation in this case results in poor concurrency.

You may use the database system monitor to help you track and tune this configuration parameter.

Related reference:

- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438

Lock Timeout configuration parameter - locktimeout

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	-1 [-1; 0 – 30 000]
Unit of Measure	Seconds

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 Data Links Manager, if you see lock timeouts in the administration notification log of the Data Links 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.

High values of the *lock_timeout* (number of lock timeouts) monitor element can be caused by:
- Too low a value for this configuration parameter.

- An application (transaction) that is holding locks 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 row-level to a table-level lock).

Related reference:

- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
- "Number of Lock Timeouts" in the *System Monitor Guide and Reference*

I/O and Storage

The following parameters can influence I/O and storage costs related to the operation of your database:

- "Changed Pages Threshold configuration parameter - chngpgs_thresh"
- "Number of Asynchronous Page Cleaners configuration parameter - num_iocleaners" on page 432
- "Number of I/O Servers configuration parameter - num_ioservers" on page 434
- "Sequential Detection Flag configuration parameter - seqdetect" on page 434
- "Default Prefetch Size configuration parameter - dft_prefetch_sz" on page 435
- "Default Number of SMS Containers configuration parameter - numsegs" on page 436
- "Default Extent Size of Table Spaces configuration parameter - dft_extent_sz" on page 436
- "Extended Storage Memory Segment Size configuration parameter - estore_seg_sz" on page 437
- "Number of Extended Storage Memory Segments configuration parameter - num_estore_segs" on page 437

Changed Pages Threshold configuration parameter - chngpgs_thresh

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	60 [5 – 99]
Unit of Measure	Percentage

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. As a result, database agents should not have to wait for changed pages to be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

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.

Related reference:

- "Number of Asynchronous Page Cleaners configuration parameter - num_iocleaners" on page 432

Number of Asynchronous Page Cleaners configuration parameter - num_iocleaners

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	1 [0 – 255]
Unit of Measure	Counter

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. As a result, database agents should not have to wait for changed pages to be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

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

  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*.

Related reference:

- "Changed Pages Threshold configuration parameter - chngpgs_thresh" on page 431
- "Buffer Pool Data Writes" in the *System Monitor Guide and Reference*
- "Buffer Pool Index Writes" in the *System Monitor Guide and Reference*
- "Buffer Pool Asynchronous Data Writes" in the *System Monitor Guide and Reference*
- "Buffer Pool Asynchronous Index Writes" in the *System Monitor Guide and Reference*

Number of I/O Servers configuration parameter - num_ioservers

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	3 [1 – 255]
Unit of Measure	Counter
When Allocated	When an application connects to a database
When Freed	When an application disconnects from a database

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.

Related reference:

- "Default Prefetch Size configuration parameter - dft_prefetch_sz" on page 435
- "Sequential Detection Flag configuration parameter - seqdetect" on page 434

Sequential Detection Flag configuration parameter - seqdetect

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	Yes [Yes; No]

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.

Related reference:

- "Default Prefetch Size configuration parameter - dft_prefetch_sz" on page 435

Default Prefetch Size configuration parameter - dft_prefetch_sz

Configuration Type	Database	
Parameter Type	Configurable Online	
Propagation Class	Immediate	
Default [Range]		
	UNIX	32 [0 — 32 767]
	Windows	16 [0 — 32 767]
Unit of Measure	Pages	

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.

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 fraction 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*).

Related reference:

- "Default Extent Size of Table Spaces configuration parameter - dft_extent_sz" on page 436
- "Number of I/O Servers configuration parameter - num_ioservers" on page 434
- "CREATE TABLESPACE statement" in the *SQL Reference, Volume 2*

Default Number of SMS Containers configuration parameter - 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.

Default Extent Size of Table Spaces configuration parameter - dft_extent_sz

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	32 [2 – 256]
Unit of Measure	Pages

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.

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.

Related concepts:

- "Extent size" in the *Administration Guide: Planning*

Related reference:

- "Default Prefetch Size configuration parameter - dft_prefetch_sz" on page 435
- "CREATE TABLESPACE statement" in the *SQL Reference, Volume 2*

Extended Storage Memory Segment Size configuration parameter - estore_seg_sz

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	16 000 [0 – 1 048 575]
Unit of Measure	Pages (4 KB)

This parameter specifies the number of pages in each of the extended memory segments in the database. This parameter is only used if your machine has more real addressable memory than the maximum amount of virtual addressable memory.

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.

Related reference:

- "Number of Extended Storage Memory Segments configuration parameter - num_estore_segs" on page 437

Number of Extended Storage Memory Segments configuration parameter - num_estore_segs

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	0 [0 – 2 147 483 647]

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.

Related reference:

- "Extended Storage Memory Segment Size configuration parameter - estore_seg_sz" on page 437
- "ALTER BUFFERPOOL statement" in the *SQL Reference, Volume 2*
- "CREATE BUFFERPOOL statement" in the *SQL Reference, Volume 2*

Agents

The following parameters can influence the number of applications that can be run concurrently and achieve optimal performance:

- "Maximum Number of Active Applications configuration parameter - maxappls"
- "Average Number of Active Applications configuration parameter - avg_appls" on page 440
- "Maximum Database Files Open per Application configuration parameter - maxfilop" on page 441
- "Maximum Total Files Open configuration parameter - maxtotfilop" on page 442
- "Priority of Agents configuration parameter - agentpri" on page 443
- "Maximum Number of Agents configuration parameter - maxagents" on page 444
- "Maximum Number of Concurrent Agents configuration parameter - maxcagents" on page 445
- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Maximum Number of Client Connections configuration parameter - max_connections" on page 448
- "Agent Pool Size configuration parameter - num_poolagents" on page 448
- "Initial Number of Agents in Pool configuration parameter - num_initagents" on page 450

Maximum Number of Active Applications configuration parameter - maxappls

Configuration Type Database

Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Automatic [Automatic; 1 – 60 000]
Unit of Measure	Counter

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.

Setting *maxappls* to *automatic* has the effect of allowing any number of connected applications. DB2 will dynamically allocate the resources it needs to support new applications.

If you do not want to set this parameter to *automatic*, 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.

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 Data Links 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 Data Links Manager)
```

The maximum supported value for Data Links 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.

Related tasks:

- "Manually resolving indoubt transactions" in the *Administration Guide: Planning*

Related reference:

- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Maximum Number of Agents configuration parameter - maxagents" on page 444
- "Maximum Storage for Lock List configuration parameter - locklist" on page 397
- "Maximum Percent of Lock List Before Escalation configuration parameter - maxlocks" on page 428
- "Average Number of Active Applications configuration parameter - avg_appls" on page 440

Average Number of Active Applications configuration parameter - avg_appls

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Statement boundary
Default [Range]	1 [1 – maxappls]
Unit of Measure	Counter

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.

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 complex 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 complex 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.

Related reference:

- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438

Maximum Database Files Open per Application configuration parameter - maxfilop

Configuration Type	Database	
Parameter Type	Configurable Online	
Propagation Class	Transaction boundary	
Default [Range]		
	UNIX	64 [2 – 1950]
	Windows	64 [2 – 32 768]
Unit of Measure	Counter	

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.

Related reference:

- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438
- "Maximum Total Files Open configuration parameter - maxtotfilop" on page 442

Maximum Total Files Open configuration parameter - maxtotfilop

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	16 000 [100 – 32 768]
Unit of Measure	Counter

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.

Related reference:

- "Maximum Database Files Open per Application configuration parameter - maxfilop" on page 441

Priority of Agents configuration parameter - agentpri

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

AIX -1 [41 - 125]

Other UNIX
-1 [41 - 128]

Windows
-1 [0 - 6]

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 subagents, 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.

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 configuration parameter - maxagents

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	200 [1 – 64 000] 400 [1 – 64 000] on Partitioned database server with local and remote clients
Unit of Measure	Counter

This parameter indicates the maximum number of database manager agents, whether coordinator 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.

Related reference:

- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Agent Pool Size configuration parameter - num_poolagents" on page 448
- "Maximum Number of Concurrent Agents configuration parameter - maxcagents" on page 445
- "Maximum Number of Fenced Processes configuration parameter - fenced_pool" on page 452
- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438
- "Minimum Committed Private Memory configuration parameter - min_priv_mem" on page 414

Maximum Number of Concurrent Agents configuration parameter - maxcagents

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	-1 (*max_coordagents*) [-1; 1 – *max_coordagents*]
Unit of Measure	Counter

The maximum number of database manager 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 the performance of the database.

Related reference:

- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Maximum Number of Agents configuration parameter - maxagents" on page 444
- "Maximum Number of Active Applications configuration parameter - maxappls" on page 438

Maximum Number of Coordinating Agents configuration parameter - max_coordagents

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	-1 (*maxagents – num_initagents*) [-1, 0 – *maxagents*]

For partitioned database environments and environments in which *intra_parallel* is set to Yes, the default is *maxagents* minus *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*.

When the Concentrator is off, that is, when *max_connections* is equal to *max_coordagents*, 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.

When the Concentrator is on, that is, when *max_connections* is greater than *max_coordagents*, there may be more connections than coordinator agents to service them. An application is in an active state only if there is a coordinator agent servicing it. Otherwise, the application is in an inactive state. Requests from an active application will be serviced by the database coordinator agent (and subagents in SMP or MPP configurations). Requests from an inactive application will be queued until a database coordinator agent is assigned to service the application, when the application becomes active. As a result, this parameter can be used to control the load on the system.

Related reference:

- "Initial Number of Agents in Pool configuration parameter - num_initagents" on page 450
- "Agent Pool Size configuration parameter - num_poolagents" on page 448
- "Enable Intra-Partition Parallelism configuration parameter - intra_parallel" on page 506
- "Maximum Number of Agents configuration parameter - maxagents" on page 444

Maximum Number of Client Connections configuration parameter - max_connections

Configuration Type Database manager

Parameter Type Configurable

Default [Range] -1 (*max_coordagents*) [-1; *max_coordagents* — 64 000]

When the Concentrator is off, this parameter indicates the maximum number of client connections allowed per partition. The Concentrator is off when *max_connections* is equal to *max_coordagents*. The Concentrator is on when *max_connections* is greater than *max_coordagents*.

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*.

In previous versions of DB2, this parameter was called *max_logicagents*.

Agent Pool Size configuration parameter - num_poolagents

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] -1 [-1, 0 — *maxagents*]

Using the default, the value for a server with a non-partitioned database and local clients is the larger of *maxagents* / 50 or *max_querydegree*.

Using the default, the value for a server with a non-partitioned database and local and remote clients is the larger of *maxagents* / 50 x *max_querydegree* or *maxagents* - *max_coordagents*.

Using the default, the value for a database partition server is the larger of *maxagents* / 10 x *max_querydegree* or *maxagents* - *max_coordagents*.

When the Concentrator is off, that is, when *max_connections* is equal to *max_coordagents*, this parameter determines the maximum size of the idle agent pool. Idle agents can be used as parallel subagents or as coordinator 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.

When the Concentrator is on, that is, when *max_connections* is greater than *max_coordagents*, this parameter will be used as a guideline for how large the agent pool will be when the system work load is low. A database agent will always be returned to the pool, no matter what the value of this parameter is. Based on the system load and the time agents remain idle in the pool, the logical agent scheduler may terminate as many of them as necessary to reduce the size of the idle pool to this parameter value.

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 coordinator 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.

Related reference:

- "Initial Number of Agents in Pool configuration parameter - num_initagents" on page 450
- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446

- "Maximum Query Degree of Parallelism configuration parameter - max_querydegree" on page 505
- "Maximum Number of Agents configuration parameter - maxagents" on page 444

Initial Number of Agents in Pool configuration parameter - num_initagents

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 0 [0 — *num_poolagents*]

This parameter determines the initial number of idle agents that are created in the agent pool at DB2START time.

Related reference:

- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Agent Pool Size configuration parameter - num_poolagents" on page 448
- "Maximum Number of Agents configuration parameter - maxagents" on page 444

Stored Procedures and User Defined Functions

The following parameters can affect fenced stored procedure and user defined function performance:

- "Keep Fenced Process configuration parameter - keepfenced"
- "Maximum Number of Fenced Processes configuration parameter - fenced_pool" on page 452
- "Initial Number of Fenced Processes configuration parameter - num_initfenced" on page 453

Keep Fenced Process configuration parameter - keepfenced

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Yes [Yes; No]

This parameter indicates whether or not a fenced mode process is kept after a fenced mode routine call is complete. Fenced mode processes are created as separate system entities in order to isolate user-written fenced mode code from the database manager agent process. This parameter is only applicable on database servers.

If *keepfenced* is set to *no*, and the routine being executed is not threadsafe, a new fenced mode process is created and destroyed for each fenced mode invocation. If *keepfenced* is set to *no*, and the routine being executed is threadsafe, the fenced mode process persists, but the thread created for the call is terminated. If *keepfenced* is set to *yes*, a fenced mode process or thread is reused for subsequent fenced mode calls. When the database manager is stopped, all outstanding fenced mode processes and threads will be terminated.

Setting this parameter to *yes* will result in additional system resources being consumed by the database manager for each fenced mode process that is activated, up to the value contained in the *fenced_pool* parameter. A new process is only created when no existing fenced mode process is available to process a subsequent fenced routine invocation. This parameter is ignored if *fenced_pool* is set to 0.

Recommendation: In an environment in which the number of fenced mode requests is large relative to the number of non-fenced mode requests, and system resources are not constrained, then this parameter can be set to *yes*. This will improve the fenced mode process performance by avoiding the initial fenced mode process creation overhead since an existing fenced mode process will be used to process the call. In particular, for Java routines, this will save the cost of starting the Java Virtual Machine (JVM), a very significant performance improvement.

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 fenced mode process. In this application, the main workload is performed out of fenced mode processes. If this parameter is set to *no*, each transaction incurs the overhead of creating a new fenced mode process, resulting in a significant performance reduction. If, however, this parameter is set to *yes*, each transaction would try to use an existing fenced mode process, which would avoid this overhead.

In previous versions of DB2, this parameter was known as *keepdari*.

Related reference:

- "Maximum Number of Fenced Processes configuration parameter - fenced_pool" on page 452

Maximum Number of Fenced Processes configuration parameter - fenced_pool

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	-1 (*max_coordagents*)
Unit of Measure	Counter

For threaded db2fmp processes (processes serving threadsafe stored procedures and UDFs), this parameter represents the number of threads cached in each db2fmp process. For nonthreaded db2fmp processes, this parameter represents the number of processes cached.

Recommendation: If your environment uses fenced stored procedures or user defined functions, then this parameter can be used to ensure that an appropriate number of db2fmp processes are available to process the maximum number of concurrent stored procedures and UDFs that run on the instance, ensuring that no new fenced mode processes need to be created as part of stored procedure and UDF execution.

If the parameter is set to −1, the maximum number of cached db2fmp processes will be the same as the value set in the *max_coordagents* parameter.

If you find that the default value is not appropriate for your environment because an inappropriate amount of system resource is being given to db2fmp processes and is affecting performance of the database manager, the following may be useful in providing a starting point for tuning this parameter:

```
fenced_pool = # of applications allowed to make stored procedure and
UDF calls at one time
```

If *keepfenced* is set to *yes*, then each db2fmp process that is created in the cache pool will continue to exist and use system resources even after the fenced routine call has been processed and returned to the agent.

If *keepfenced* is set to *no*, then nonthreaded db2fmp processes will terminate when they complete execution, and there is no cache pool. Multithreaded db2fmp processes will continue to exist, but no threads will be pooled in these processes. This means that even when *keepfenced* is set *no* you can have one threaded C db2fmp process and one threaded Java db2fmp process on your system.

In previous versions of DB2, this parameter was known as *maxdari*.

Related reference:

- "Maximum Number of Coordinating Agents configuration parameter - max_coordagents" on page 446
- "Maximum Number of Agents configuration parameter - maxagents" on page 444
- "Keep Fenced Process configuration parameter - keepfenced" on page 450
- "Initial Number of Fenced Processes configuration parameter - num_initfenced" on page 453

Initial Number of Fenced Processes configuration parameter - num_initfenced

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	0 [0 — *max_connections* + (*maxagents* - *max_coordagents*)]

This parameter indicates the initial number of nonthreaded, idle db2fmp processes that are created in the db2fmp pool at DB2START time. Setting this parameter will reduce the initial startup time for running non-threadsafe C and Cobol routines. This parameter is ignored if *keepfenced* is not specified.

It is much more important to set *fenced_pool* to an appropriate size for your system than to start up a number of db2fmp processes at DB2START time.

In previous versions of DB2, this parameter was known as *num_initdaris*.

Related reference:

- "Keep Fenced Process configuration parameter - keepfenced" on page 450

- “Maximum Number of Fenced Processes configuration parameter - fenced_pool” on page 452

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 464
- “Recovery” on page 469
- “Distributed Unit of Work Recovery” on page 476.

Database Log Files

The following parameters provide information about number, size and status of the files used for database logging:

- “Size of Log Files configuration parameter - logfilsiz”
- “Number of Primary Log Files configuration parameter - logprimary” on page 456
- “Number of Secondary Log Files configuration parameter - logsecond” on page 458
- “Change the Database Log Path configuration parameter - newlogpath” on page 459
- “Mirror Log Path configuration parameter - mirrorlogpath” on page 461
- “Overflow Log Path configuration parameter - overflowlogpath” on page 462
- “Location of Log Files configuration parameter - logpath” on page 463
- “First Active Log File configuration parameter - loghead” on page 464

Size of Log Files configuration parameter - logfilsiz

Configuration Type	Database	
Parameter Type	Configurable	
Default [Range]		
	UNIX	1000 [4 — 262 144]
	Windows	250 [4 — 262 144]
Unit of Measure	Pages (4 KB)	

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 upper limit of log file size, combined with the upper limit of the number of log files (*logprimary* + *logsecond*), gives an upper limit of 256 GB of active log space.

  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.

Related reference:

- "Number of Primary Log Files configuration parameter - logprimary" on page 456
- "Number of Secondary Log Files configuration parameter - logsecond" on page 458
- "Recovery Range and Soft Checkpoint Interval configuration parameter - softmax" on page 465

Number of Primary Log Files configuration parameter - logprimary

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	3 [2 – 256]
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.

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*) <= 256

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. Observation of the following monitor values over a period of time will aid in better tuning decisions, as average values may be more representative of your ongoing requirements.

- *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)*

Related reference:

- “Size of Log Files configuration parameter - logfilsiz” on page 454
- “Number of Secondary Log Files configuration parameter - logsecond” on page 458
- “Log Retain Enable configuration parameter - logretain” on page 467
- “User Exit Enable configuration parameter - userexit” on page 468
- “Maximum Secondary Log Space Used” in the *System Monitor Guide and Reference*
- “Maximum Total Log Space Used” in the *System Monitor Guide and Reference*

- "Secondary Logs Allocated Currently" in the *System Monitor Guide and Reference*

Number of Secondary Log Files configuration parameter - logsecond

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	2 [-1; 0 – 254]
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.

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 shut down, if more secondary log files are required than are allowed by this parameter.

If you set *logsecond* to -1, the database is configured with infinite active log space. There is no limit on the size or the number of in-flight transactions running on the database. If you set *logsecond* to -1, you still use the *logprimary* and *logfilsiz* configuration parameters to specify how many log files DB2 should keep in the active log path. If DB2 needs to read log data from a log file, but the file is not in the active log path, DB2 will invoke the userexit program to retrieve the log file from the archive to the active log path. (DB2 will retrieve the files to the overflow log path, if you have configured one.) Once the log file is retrieved, DB2 will cache this file in the active log path so that other reads of log data from the same file will be fast. DB2 will manage the retrieval, caching, and removal of these log files as required.

If your log path is a raw device, you must configure the *overflowlogpath* configuration parameter in order to set *logsecond* to -1.

By setting *logsecond* to -1, you will have no limit on the size of the unit of work or the number of concurrent units of work. However, rollback (both at the savepoint level and at the unit of work level) could be very slow due to the need to retrieve log files from the archive. Crash recovery could also be very slow for the same reason. DB2 will write a message to the administration

notification log to warn you that the current set of active units of work has exceeded the primary log files. This is an indication that rollback or crash recovery could be extremely slow.

To set *logsecond* to -1 the *userexit* configuration parameter must be set to *yes*.

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.

Related reference:

- "Size of Log Files configuration parameter - logfilsiz" on page 454
- "Number of Primary Log Files configuration parameter - logprimary" on page 456
- "Log Retain Enable configuration parameter - logretain" on page 467
- "User Exit Enable configuration parameter - userexit" on page 468
- "Overflow Log Path configuration parameter - overflowlogpath" on page 462

Change the Database Log Path configuration parameter - newlogpath

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any valid path or device]

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.

If you configure the *mirrorlogpath* configuration parameter, you must specify a path name for *newlogpath*, not a raw device.

If you want to use replication, and your log path is a raw device, the *overflowlogpath* configuration parameter must be configured.

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 or later 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 2000, Windows NT, Solaris Operating Environment, HP-UX, and Linux 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.

The monitor elements *log_reads* (number of log pages read) and *log_writes* (number of log pages written) 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.

Related reference:

- "Location of Log Files configuration parameter - logpath" on page 463

- “Database is Consistent configuration parameter - database_consistent” on page 487
- “Number of Log Pages Read” in the *System Monitor Guide and Reference*
- “Number of Log Pages Written” in the *System Monitor Guide and Reference*

Mirror Log Path configuration parameter - mirrorlogpath

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any valid path or device]

This parameter allows you to specify a string of up to 242 bytes for the mirror log path. The string must point to a path name, and it must be a fully qualified path name, not a relative path name. If you configure the *mirrorlogpath* parameter, you must specify a path name for *newlogpath*, not a raw device.

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.

If *mirrorlogpath* is configured, DB2 will create active log files in both the log path and the mirror log path. All log data will be written to both paths. The mirror log path has a duplicated set of active log files, such that if there is a disk error or human error that destroys active log files on one of the paths, the database can still function.

If the mirror log path is changed, there might be log files in the old mirror log path. These log files might not have been archived, so 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 mirror log path to the new mirror log path.

Recommendation: Just like the log files, the mirror log files should be on a physical disk that does not have high I/O.

It is strongly recommended that this path be on a separate device than the primary log path.

You may use the database system monitor to track the number of I/O's related to database logging.

The following 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.

- *log_reads* (number of log pages read)
- *log_writes* (number of log pages written).

Related reference:

- "Location of Log Files configuration parameter - logpath" on page 463
- "Change the Database Log Path configuration parameter - newlogpath" on page 459
- "Number of Log Pages Read" in the *System Monitor Guide and Reference*
- "Number of Log Pages Written" in the *System Monitor Guide and Reference*
- "Overflow Log Path configuration parameter - overflowlogpath" on page 462

Overflow Log Path configuration parameter - overflowlogpath

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	Null [any valid path]

This parameter can be used for several functions, depending on your logging requirements.

- This parameter allows you to specify a location for DB2 to find log files that are needed for a rollforward operation. It is similar to the OVERFLOW LOG PATH option on the ROLLFORWARD command. Instead of always specifying OVERFLOW LOG PATH on every ROLLFORWARD command, you can set this configuration parameter once. However, if both are used, the OVERFLOW LOG PATH option will overwrite the *overflowlogpath* configuration parameter, for that particular rollforward operation.
- If *logsecond* is set to -1, *overflowlogpath* allows you to specify a directory for DB2 to store active log files retrieved from the archive. (Active log files have to be retrieved for rollback operations if they are no longer in the active log path). Without overflowlogpath, DB2 will retrieve the log files into the active log path. Using *overflowlogpath* allows you to provide additional resource for DB2 to store the retrieved log files. The benefit includes spreading the I/O cost to different disks, and allowing more log files to be stored in the active log path.

- If you need to use the db2ReadLog API (prior to DB2 V8, db2ReadLog was called sqlurlog) for replication, for example, *overflowlogpath* allows you to specify a location for DB2 to search for log files that are needed for this API. If the log file is not found (in either the active log path or the overflow log path) and the database is configured with *userexit* enabled, DB2 will retrieve the log file. *overflowlogpath* also allows you to specify a directory for DB2 to store the log files retrieved. The benefit comes from reducing the I/O cost on the active log path and allowing more log files to be stored in the active log path.
- If you have configured a raw device for the active log path, *overflowlogpath* must be configured if you want to set *logsecond* to -1, or if you want to use the db2ReadLog API.

To set *overflowlogpath*, specify a string of up to 242 bytes. The string must point to a path name, and it must be a fully qualified path name, not a relative path name. The path name must be a directory, not a raw device.

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.

Related reference:

- "Number of Secondary Log Files configuration parameter - logsecond" on page 458
- "db2ReadLog - Asynchronous Read Log" in the *Administrative API Reference*
- "ROLLFORWARD DATABASE Command" in the *Command Reference*

Location of Log Files configuration parameter - logpath

Configuration Type	Database
Parameter Type	Informational

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.

Related reference:

- "Change the Database Log Path configuration parameter - newlogpath" on page 459

First Active Log File configuration parameter - 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 configuration parameter - mincommit"
- "Recovery Range and Soft Checkpoint Interval configuration parameter - softmax" on page 465
- "Log Retain Enable configuration parameter - logretain" on page 467
- "User Exit Enable configuration parameter - userexit" on page 468
- "Block on Log Disk Full configuration parameter - blk_log_dsk_ful" on page 469

Number of Commits to Group configuration parameter - mincommit

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
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. As a result, this will 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 could be 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 transaction intensive 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 transaction intensive 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 transaction intensive 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)*

Related reference:

- "Unit of Work Log Space Used" in the *System Monitor Guide and Reference*
- "Commit Statements Attempted" in the *System Monitor Guide and Reference*
- "Rollback Statements Attempted" in the *System Monitor Guide and Reference*

Recovery Range and Soft Checkpoint Interval configuration parameter - softmax

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	100 [1 – 100 * *logprimary*]
Unit of Measure	Percentage of the size of one primary log file

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
```

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.)

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.

Related reference:

- "Size of Log Files configuration parameter - logfilsiz" on page 454
- "Number of Primary Log Files configuration parameter - logprimary" on page 456

Log Retain Enable configuration parameter - logretain

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Recovery; No]

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.

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 because 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.

Related reference:

- "Log Retain Status Indicator configuration parameter - log_retain_status" on page 488
- "User Exit Enable configuration parameter - userexit" on page 468
- "Backup Pending Indicator configuration parameter - backup_pending" on page 487

User Exit Enable configuration parameter - userexit

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Yes; No]

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.

Related reference:

- "User Exit for Database Recovery" in the *Data Recovery and High Availability Guide and Reference*
- "Log Retain Enable configuration parameter - logretain" on page 467
- "User Exit Status Indicator configuration parameter - user_exit_status" on page 488
- "Backup Pending Indicator configuration parameter - backup_pending" on page 487
- "ROLLFORWARD DATABASE Command" in the *Command Reference*

Block on Log Disk Full configuration parameter - blk_log_dsk_ful

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	No [Yes; No]

This configuration parameter can be set to prevent disk full errors from being generated when DB2 cannot create a new log file in the active log path. Instead, DB2 will attempt to create the log file every five minutes until it succeeds. After each attempt, DB2 writes a message to the Administration Notification log. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the Administration Notification log. Until the log file is successfully created, any user application that attempts to update table data will not be able to commit transactions. Read-only queries may not be directly affected; however, if a query needs to access data that is locked by an update request, or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Setting *blk_log_dsk_ful* to *yes* causes applications to hang when DB2 encounters a log disk full error, thus allowing you to resolve the error and allowing the transaction to complete. You can resolve a disk full situation by moving old log files to another file system or by enlarging the file system, so that hanging applications can complete.

If *blk_log_dsk_ful* is set to *no*, then a transaction that receives a log disk full error will fail and will be rolled back. In some situations, the database will come down if a transaction causes a log disk full error.

Recovery

The following parameters affect various aspects of database recovery:

- “Auto Restart Enable configuration parameter - autorestart” on page 470
- “Index Re-creation Time configuration parameter - indexrec” on page 470
- “Default Number of Load Recovery Sessions configuration parameter - dft_loadrec_ses” on page 472
- “Number of Database Backups configuration parameter - num_db_backups” on page 473
- “Recovery History Retention Period configuration parameter - rec_his_retentn” on page 473
- “Track Modified Pages Enable configuration parameter - trackmod” on page 474

See also “Distributed Unit of Work Recovery” on page 476.

The following parameters are used when working with Tivoli Storage Manager (TSM):

- "Tivoli Storage Manager Management Class configuration parameter - tsm_mgmtclass" on page 474
- "Tivoli Storage Manager Password configuration parameter - tsm_password" on page 475
- "Tivoli Storage Manager Node Name configuration parameter - tsm_nodename" on page 475
- "Tivoli Storage Manager Owner Name configuration parameter - tsm_owner" on page 476

Auto Restart Enable configuration parameter - autorestart

Configuration Type Database

Parameter Type Configurable Online

Propagation Class Immediate

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.

Related concepts:

- "Crash Recovery" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

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

Index Re-creation Time configuration parameter - indexrec

Configuration Type Database and Database Manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

UNIX Database Manager
restart [restart; access]

Windows Database Manager
access [restart; access]

Database Use system setting [system; restart; access]

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.

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 index is recreated.

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.

Related reference:

- "Auto Restart Enable configuration parameter - autorestart" on page 470
- "RESTART DATABASE Command" in the *Command Reference*

Default Number of Load Recovery Sessions configuration parameter - dft_loadrec_ses

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	1 [1 – 30 000]
Unit of Measure	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.

Related concepts:

- "Load Overview" in the *Data Movement Utilities Guide and Reference*

Related reference:

- "Miscellaneous variables" on page 571

Number of Database Backups configuration parameter - num_db_backups

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Transaction boundary
Default [Range]	12 [1 — 32 768]

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, TSM). 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.

Related reference:

- "Recovery History Retention Period configuration parameter - rec_his_retentn" on page 473

Recovery History Retention Period configuration parameter - rec_his_retentn

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	366 [-1; 0 — 30 000]
Unit of Measure	Days

This parameter is used to specify the number of days that historical information on backups should be retained. If the recovery history file is not needed to keep track of backups, restores, and loads, this parameter can be set to a small number.

If value of this parameter is -1, the recovery history file can only be pruned explicitly using the available commands or APIs. If the value is not -1, the recovery history file is pruned after every full database backup.

The 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.

Related reference:

- "PRUNE HISTORY/LOGFILE Command" in the *Command Reference*
- "Number of Database Backups configuration parameter - num_db_backups" on page 473

Track Modified Pages Enable configuration parameter - trackmod

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Yes, No]

When this parameter is set to ″Yes″, the database manager tracks database modifications so that the backup utility can detect which subsets of the database pages must be examined by an incremental backup and potentially included in the backup image. After setting this parameter to ″Yes″, you must take a full database backup in order to have a baseline against which incremental backups can be taken. Also, if this parameter is enabled and if a table space is created, then a backup must be taken which contains that table space. This backup could be either a database backup or a table space backup. Following the backup, incremental backups will be permitted to contain this table space.

Tivoli Storage Manager Management Class configuration parameter - tsm_mgmtclass

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Statement boundary
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.

Tivoli Storage Manager Password configuration parameter - tsm_password

Configuration Type Database

Parameter Type Configurable online

Propagation Class Statement boundary

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.

Tivoli Storage Manager Node Name configuration parameter - tsm_nodename

Configuration Type Database

Parameter Type Configurable online

Propagation Class Statement boundary

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).

Tivoli Storage Manager Owner Name configuration parameter - tsm_owner

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Statement boundary
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 TSM 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.

Distributed Unit of Work Recovery

The following parameters affect the recovery of distributed unit of work (DUOW) transactions:

- "Transaction Manager Database Name configuration parameter - tm_database"
- "Transaction Resync Interval configuration parameter - resync_interval" on page 477
- "Sync Point Manager Log File Path configuration parameter - spm_log_path" on page 478
- "Sync Point Manager Name configuration parameter - spm_name" on page 478
- "Sync Point Manager Log File Size configuration parameter - spm_log_file_sz" on page 479
- "Sync Point Manager Resync Agent Limit configuration parameter - spm_max_resync" on page 479

Transaction Manager Database Name configuration parameter - tm_database

Configuration Type	Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 1ST_CONN [any valid database name]

This parameter identifies 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 (SPM) 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.

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 configuration parameter - resync_interval

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 180 [1 – 60 000]

Unit of Measure 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. This parameter also applies to recovery of federated database systems.

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 configuration parameter - 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

Parameter Type Configurable

Default [Range] 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.

Sync Point Manager Name configuration parameter - spm_name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Derived from the TCP/IP hostname

This parameter identifies the name of the sync point manager (SPM) instance to the database manager.

Sync Point Manager Log File Size configuration parameter - spm_log_file_sz

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 256 [4 — 1 000]

Unit of Measure Pages (4 KB)

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.

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 configuration parameter - spm_max_resync

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 20 [10 — 256]

This parameter identifies the number of agents that can simultaneously perform resync operations.

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"
- "Attributes" on page 481
- "DB2 Data Links Manager" on page 483
- "Status" on page 486
- "Compiler Settings" on page 489.

Query Enabler

The following parameters provide information for the control of Query Enabler:

- "Dynamic SQL Query Management configuration parameter - dyn_query_mgmt"

Dynamic SQL Query Management configuration parameter - 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 configuration parameter - release”
- “Database Release Level configuration parameter - database_level”
- “Database Territory parameter - territory” on page 482
- “Database Territory Code configuration parameter” on page 482
- “Codeset for the Database configuration parameter - codeset” on page 482
- “Code Page for the Database configuration parameter - codepage” on page 482
- “Collating Information configuration parameter - collate_info” on page 483

With the exception of *copyprotect*, these parameters are provided for informational purposes only.

Configuration File Release Level configuration parameter - release

Configuration Type Database manager, Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Informational

This parameter specifies the release level of the configuration file.

Related reference:

- “Database Release Level configuration parameter - database_level” on page 481

Database Release Level configuration parameter - database_level

Configuration Type Database

Parameter Type Informational

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.

Related reference:

- "Configuration File Release Level configuration parameter - release" on page 481

Database Territory parameter - territory

Configuration Type Database

Parameter Type Informational

This parameter shows the territory used to create the database. *territory* is used by the database manager to determine the territory code (*territory*) parameter values.

Related reference:

- "Database Territory Code configuration parameter" on page 482

Database Territory Code configuration parameter

Configuration Type Database

Parameter Type Informational

This parameter shows the territory code used to create the database.

Related reference:

- "Database Territory parameter - territory" on page 482

Codeset for the Database configuration parameter - codeset

Configuration Type Database

Parameter Type Informational

This parameter shows the codeset that was used to create the database. Codeset is used by the database manager to determine *codepage* parameter values.

Related reference:

- "Code Page for the Database configuration parameter - codepage" on page 482

Code Page for the Database configuration parameter - codepage

Configuration Type Database

Parameter Type Informational

This parameter shows the code page that was used to create the database. The *codepage* parameter is derived based on the *codeset* parameter.

Related reference:

- "Codeset for the Database configuration parameter - codeset" on page 482

Collating Information configuration parameter - 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.

DB2 Data Links Manager

The following parameters relate to DB2 Data Links Manager:

- "Data Links Access Token Expiry Interval configuration parameter - dl_expint" on page 484
- "Data Links Write Token Initial Expiry Interval configuration parameter - dl_wt_iexpint" on page 484
- "Data Links Number of Copies configuration parameter - dl_num_copies" on page 485
- "Data Links Time After Drop configuration parameter - dl_time_drop" on page 485

- “Data Links Token Algorithm configuration parameter - dl_token” on page 485
- “Data Links Token in Upper Case configuration parameter - dl_upper” on page 486
- “Enable Data Links Support configuration parameter - datalinks” on page 486

Data Links Access Token Expiry Interval configuration parameter - dl_expint

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Transaction boundary
Default [Range]	60 [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.

This parameter applies to the DATALINK columns that specify “READ PERMISSION DB”.

Data Links Write Token Initial Expiry Interval configuration parameter - dl_wt_iexpint

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Transaction boundary
Default [Range]	60 [1 – 31 536 000]
Unit of Measure	Seconds

This parameter specifies the initial expiry interval of a write token generated from a DATALINK column. The initial expiry interval is the period between the time a token is generated and the first time it is used to open the file.

This parameter only applies to a DATALINK column defined with WRITE PERMISSION ADMIN.

Recommendation: The value should be large enough to cover the time from when a write token is retrieved to when it is used to open the file.

Data Links Number of Copies configuration parameter - dl_num_copies

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
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 a TSM 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 configuration parameter - dl_time_drop

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
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 a 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 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 configuration parameter - dl_token

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Transaction boundary
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.

This parameter applies to the DATALINK columns that specify “READ PERMISSION DB”.

Data Links Token in Upper Case configuration parameter - dl_upper

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Transaction boundary
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.

This parameter applies to the DATALINK columns that specify “READ PERMISSION DB”.

Enable Data Links Support configuration parameter - 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 configuration parameter - backup_pending” on page 487
- “Database is Consistent configuration parameter - database_consistent” on page 487
- “Roll Forward Pending Indicator configuration parameter - rollfwd_pending” on page 487
- “Log Retain Status Indicator configuration parameter - log_retain_status” on page 488

- “User Exit Status Indicator configuration parameter - user_exit_status” on page 488
- “Restore Pending configuration parameter - restore_pending” on page 488
- “Multipage File Allocation Enabled configuration parameter - multipage_alloc” on page 488

Backup Pending Indicator configuration parameter - 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 configuration parameter - 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.

Roll Forward Pending Indicator configuration parameter - 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 spaces need 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.

Log Retain Status Indicator configuration parameter - log_retain_status

Configuration Type Database

Parameter Type Informational

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 is equal to Recovery.

Related reference:

- "Log Retain Enable configuration parameter - logretain" on page 467

User Exit Status Indicator configuration parameter - user_exit_status

Configuration Type Database

Parameter Type Informational

If set to Yes, 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.

Related reference:

- "User Exit Enable configuration parameter - userexit" on page 468

Restore Pending configuration parameter - restore_pending

Configuration Type Database

Parameter Type Informational

This parameter states whether a RESTORE PENDING status exists in the database.

Related reference:

- "User Exit Enable configuration parameter - userexit" on page 468

Multipage File Allocation Enabled configuration parameter - 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 configuration parameter - dft_sqlmathwarn"
- "Default Degree configuration parameter - dft_degree" on page 491
- "Default Query Optimization Class configuration parameter - dft_queryopt" on page 491
- "Default Refresh Age configuration parameter - dft_refresh_age" on page 492
- "Number of Frequent Values Retained configuration parameter - num_freqvalues" on page 493
- "Number of Quantiles for Columns configuration parameter - num_quantiles" on page 494

Continue upon Arithmetic Exceptions configuration parameter - 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 configuration parameter - dft_degree

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	1 [-1, 1 – 32 767]

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 an 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

Related reference:

- "Maximum Query Degree of Parallelism configuration parameter - max_querydegree" on page 505

Default Query Optimization Class configuration parameter - dft_queryopt

Configuration Type	Database

Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	5 [0 — 9]
Unit of Measure	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 option on the 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

Related reference:

- "SET CURRENT QUERY OPTIMIZATION statement" in the *SQL Reference, Volume 2*
- "BIND Command" in the *Command Reference*
- "LIST DRDA INDOUBT TRANSACTIONS Command" in the *Command Reference*

Default Refresh Age configuration parameter - 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 if 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 materialized query 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 has a

value of two, or five or more, REFRESH DEFERRED materialized query tables are considered to optimize the processing of a dynamic SQL query.

Number of Frequent Values Retained configuration parameter - num_freqvalues

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	10 [0 — 32 767]
Unit of Measure	Counter

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.

You can also specify the number of frequent values retained as part of the RUNSTATS command at the table or the column level. If none is specified, the *num_freqvalues* configuration parameter value is used.

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.

The RUNSTATS command allows for the specification of the number of frequent values retained, by using the NUM_REQVALUES option. Changing the number of frequent values retained through the RUNSTATS command is easier than making the change using the *num_freqvalues* database configuration parameter.

When using RUNSTATS, you have the ability to limit the number of frequent values collected at both the table level and the column level. This allows you

to optimize on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

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.

Related reference:

- "Number of Quantiles for Columns configuration parameter - num_quantiles" on page 494
- "Statistics Heap Size configuration parameter - stat_heap_sz" on page 410

Number of Quantiles for Columns configuration parameter - num_quantiles

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	20 [0 – 32 767]
Unit of Measure	Counter

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.

You can also specify the number of quantiles collected as part of the RUNSTATS command at the table or the column level. If none is specified, the *num_quantiles* configuration parameter value is used.

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.

The RUNSTATS command allows for the specification of the number of quantiles that will be collected, by using the NUM_QUANTILES option. Changing the number of quantiles that will be collected through the RUNSTATS command is easier than making the change using the *num_quantiles* database configuration parameter.

When using RUNSTATS, you have the ability to limit the number of quantiles collected at both the table level and the column level. This allows you to optimize on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

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 simple way to approximate 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.

Related reference:
- “Number of Frequent Values Retained configuration parameter - num_freqvalues” on page 493
- “Statistics Heap Size configuration parameter - stat_heap_sz” on page 410

Communications

The following groups of parameters provide information about using DB2 in a client/server environment:

- "Communication Protocol Setup"
- "DB2 Discovery" on page 498

Communication Protocol Setup

You can use the following parameters to configure your database clients and database servers:

- "NetBIOS Workstation Name configuration parameter - nname"
- "TCP/IP Service Name configuration parameter - svcename" on page 497
- "APPC Transaction Program Name configuration parameter - tpname" on page 498

NetBIOS Workstation Name configuration parameter - 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.

Related reference:

- "CATALOG NETBIOS NODE Command" in the *Command Reference*

TCP/IP Service Name configuration parameter - 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
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.

On UNIX-based systems, the services file is located in: `/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).

APPC Transaction Program Name configuration parameter - 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 server when using the APPC communication protocol. This parameter must be set in the configuration file at the database server.

This parameter must be the same as the transaction program name that is configured in the SNA transaction program definition.

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 (.)

DB2 Discovery

You can use the following parameters to establish DB2 Discovery:

- "Discover Database configuration parameter - discover_db"
- "Discovery Mode configuration parameter - discover" on page 499
- "Search Discovery Communications Protocols configuration parameter - discover_comm" on page 499
- "Discover Server Instance configuration parameter - discover_inst" on page 500

Discover Database configuration parameter - discover_db

Configuration Type Database

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] Enable [Disable, Enable]

This parameter is used to prevent information about a database from being returned to a client when a discovery request is received at the server.

The default for this parameter is that discovery is enabled for this database.

By changing this parameter value to “Disable”, it is possible to hide databases with sensitive data from the discovery process. This can be done in addition to other database security controls on the database.

Discovery Mode configuration parameter - discover

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] SEARCH [DISABLE, KNOWN, SEARCH]

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.

Related reference:

- “Search Discovery Communications Protocols configuration parameter - discover_comm” on page 499
- “Communications variables” on page 548

Search Discovery Communications Protocols configuration parameter - 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

Parameter Type Configurable

Default [Range] None [Any combination of NETBIOS and TCPIP]

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.

Related reference:

- “Discovery Mode configuration parameter - discover” on page 499
- “Communications variables” on page 548

Discover Server Instance configuration parameter - 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 online

Propagation Class Immediate

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.

Partitioned Database Environment

The following groups of parameters provide information about parallel operations and partitioned database environments:

- "Communications"
- "Parallel Processing" on page 505.

Communications

The following parameters provide information about communications in the partitioned database environment:

- "Connection Elapse Time configuration parameter - conn_elapse"
- "Number of FCM Buffers configuration parameter - fcm_num_buffers" on page 502
- "Node Connection Retries configuration parameter - max_connretries" on page 503
- "Maximum Time Difference Among Nodes configuration parameter - max_time_diff" on page 504
- "Start and Stop Timeout configuration parameter - start_stop_time" on page 504.

Connection Elapse Time configuration parameter - conn_elapse

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	10 [0–100]
Unit of Measure	Seconds

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.

Related reference:

- "Node Connection Retries configuration parameter - max_connretries" on page 503

Number of FCM Buffers configuration parameter - fcm_num_buffers

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

32-bit platforms
512, 1 024, or 4 096 [128 — *65 300*]

64-bit platforms
512, 1 024, or 4 096 [128 — *524 288*]

- 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, this parameter is not used if the *intra_parallel* parameter is not active.

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within database servers.

If you have multiple logical nodes on the same machine, 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 default configuration for all non-AIX platforms.

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.

Related concepts:

- "Fast communications manager (FCM) communications" in the *Administration Guide: Implementation*

Node Connection Retries configuration parameter - max_connretries

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	5 [0–100]

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.

Related reference:

- "Connection Elapse Time configuration parameter - conn_elapse" on page 501

Maximum Time Difference Among Nodes configuration parameter - 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 SQLCODE will be returned. (The transaction is rejected only if data modification is associated with it.)

DB2 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 configuration parameter - start_stop_time

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
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 ADD DBPARTITIONNUM 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.)

Related reference:

- "ADD DBPARTITIONNUM Command" in the *Command Reference*

Parallel Processing

The following parameters provide information about parallel processing:

- "Maximum Query Degree of Parallelism configuration parameter - max_querydegree"
- "Enable Intra-Partition Parallelism configuration parameter - intra_parallel" on page 506.

Maximum Query Degree of Parallelism configuration parameter - max_querydegree

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Statement boundary
Default [Range]	-1 (ANY) [ANY, 1 — 32 767] (ANY means system determined)

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.

Related reference:

- "Enable Intra-Partition Parallelism configuration parameter - intra_parallel" on page 506
- "Default Degree configuration parameter - dft_degree" on page 491

Enable Intra-Partition Parallelism configuration parameter - intra_parallel

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	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.

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.

Related reference:

- "Maximum Query Degree of Parallelism configuration parameter - max_querydegree" on page 505

Instance Management

A number of parameters can help you manage your database manager instances. These are grouped into the following categories:

- "Diagnostic"
- "Database System Monitor Parameters" on page 511
- "System Management" on page 512
- "Instance Administration" on page 520

Diagnostic

The following parameters allow you to control diagnostic information available from the database manager:

- "Diagnostic Error Capture Level configuration parameter - diaglevel"
- "Diagnostic Data Directory Path configuration parameter - diagpath" on page 508
- "Notify Level configuration parameter - notifylevel" on page 509
- "Health Monitoring configuration parameter - health_mon" on page 510

Diagnostic Error Capture Level configuration parameter - diaglevel

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] 3 [0 — 4]

The type of diagnostic errors recorded in the db2diag.log 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.

Related reference:

- "Diagnostic Data Directory Path configuration parameter - diagpath" on page 508

Diagnostic Data Directory Path configuration parameter - diagpath

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] Null [any valid path name]

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, a notification file, 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 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.

Recommendation: Use the default or have a centralized location for the diagpath of multiple instances.

In a partitioned database environment, the path you specify must reside on a shared file system.

Related reference:

- "Diagnostic Error Capture Level configuration parameter - diaglevel" on page 507

Notify Level configuration parameter - notifylevel

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 Online
Propagation Class	Immediate
Default [Range]	3 [0 — 4]

This parameter specifies the type of administration notification messages that are written to the administration notification log. On UNIX platforms, the administration notification log is a text file called *instance*`.nfy`. On Windows,

all administration notification messages are written to the Event Log. The errors can be written by DB2, the Health Monitor, the Capture and Apply programs, and user applications.

Valid values for this parameter are:

0 — No administration notification messages captured. (This setting is not recommended.)

1 — Fatal or unrecoverable errors. Only fatal and unrecoverable errors are logged. To recover from some of these conditions, you may need assistance from DB2 service.

2 — Immediate action required. Conditions are logged that require immediate attention from the system administrator or the database administrator. If the condition is not resolved, it could lead to a fatal error. Notification of very significant, non-error activities (for example, recovery) may also be logged at this level. This level will capture Health Monitor alarms.

3 — Important information, no immediate action required. Conditions are logged that are non-threatening and do not require immediate action but may indicate a non-optimal system. This level will capture Health Monitor alarms, Health Monitor warnings, and Health Monitor attentions.

4 — Informational messages.

The administration notification log includes messages having values up to and including the value of *notifylevel*. For example, setting *notifylevel* to 3 will cause the administration notification log to include messages applicable to levels 1, 2, and 3.

For a user application to be able to write to the notification file or Windows Event Log, it must call the db2AdminMsgWrite API.

Recommendation: You may wish to increase the value of this parameter to gather additional problem determination data to help resolve a problem. Note that you must set *notifylevel* to a value of 2 or higher for the Health Monitor to send any notifications to the contacts defined in its configuration.

Health Monitoring configuration parameter - health_mon

Configuration Type	Database manager
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	On [On; Off]
Related Parameters	

This parameter allows you to specify whether you want to monitor an instance, its associated databases, and database objects according to various health indicators. If *health_mon* is turned on (the default), an agent will collect information about the health of the objects you have selected. If an object is considered to be in an unhealthy position, based on thresholds that you have set, then notifications can be sent and actions can be taken automatically. If *health_mon* is turned off, then the health of objects will not be monitored.

You can use the Health Center or the CLP to select the instance and database objects that you want to monitor. You can also specify where notifications should be sent, and what actions should be taken, based on the data collected by the health monitor.

Database System Monitor Parameters

The following parameter allows you to control various aspects of the database system monitor:

- "Default Database System Monitor Switches configuration parameter - dft_monswitches"

Default Database System Monitor Switches configuration parameter - dft_monswitches

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default All switches turned off, except dft_mon_timestamp, which is turned on by default

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. You can 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
dft_mon_timestamp	Default value of the snapshot monitor's timestamp switch

Recommendation: Any switch (except dft_mon_timestamp) 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. Turning the dft_mon_timestamp switch OFF becomes important as CPU utilization approaches 100%. When this occurs, the CPU time required for issuing timestamps increases dramatically. Furthermore, if the timestamp switch is turned OFF, the overall cost of other data under monitor switch control is greatly reduced.

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.)

Related reference:

- "GET MONITOR SWITCHES Command" in the *Command Reference*

System Management

The following parameters relate to system management:

- "Communications Bandwidth configuration parameter - comm_bandwidth" on page 513
- "CPU Speed configuration parameter - cpuspeed" on page 513
- "Maximum Number of Concurrently Active Databases configuration parameter - numdb" on page 514
- "Transaction Processor Monitor Name configuration parameter - tp_mon_name" on page 516
- "Machine Node Type configuration parameter - nodetype" on page 518

- "Default Charge-Back Account configuration parameter - dft_account_str" on page 518
- "Java Development Kit Installation Path configuration parameter - jdk_path" on page 519
- "Federated Database System Support configuration parameter - federated" on page 520.

Communications Bandwidth configuration parameter - comm_bandwidth

Configuration Type	Database manager
Applies to	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Statement boundary
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 configuration parameter - cpuspeed

Configuration Type	Database manager
Applies to	

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable online

Propagation Class Statement boundary

Default [Range] -1 [1^{-10} — 1] A value of -1 will cause the parameter value to be reset based on the running of the measurement program.

Unit of Measure Seconds

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 configuration parameter - numdb

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX 8 [1 — 256]

Windows Database server with local and remote clients 8 [1 — 256]

Windows 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.

Related reference:

- “Application Control Heap Size configuration parameter - app_ctl_heap_sz” on page 403
- “Sort Heap Size configuration parameter - sortheap” on page 405
- “Application Support Layer Heap Size configuration parameter - aslheapsz” on page 417

- “Application Heap Size configuration parameter - applheapsz” on page 410
- “Maximum Storage for Lock List configuration parameter - locklist” on page 397
- “Database Heap configuration parameter - dbheap” on page 392
- “Statement Heap Size configuration parameter - stmtheap” on page 409
- “Database System Monitor Heap Size configuration parameter - mon_heap_sz” on page 422
- “Statistics Heap Size configuration parameter - stat_heap_sz” on page 410
- “Database Shared Memory Size configuration parameter - database_memory” on page 391

Transaction Processor Monitor Name configuration parameter - tp_mon_name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default No default

Valid Values

- CICS
- MQ
- ENCINA
- CB
- SF
- TUXEDO
- TOPEND
- blank or some other value (for UNIX and Windows; 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 on Windows NT, 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 and Windows NT environments.

TXSeries CICS and Encina Users: In previous versions of this product on 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 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 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 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.

Related reference:

- "xa_open string formats" in the *Administration Guide: Planning*

Machine Node Type configuration parameter - nodetype

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Informational

This parameter provides information about the DB2 products which you have installed on your machine and, as a result, information about the type of database manager configuration. The following are the possible values returned by this parameter and the products associated with that node type:

- **Database server with local and remote clients** – a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers.
- **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.

Default Charge-Back Account configuration parameter - dft_account_str

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client

- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

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 Installation Path configuration parameter - jdk_path

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [Valid path]

This parameter specifies the directory under which the Java Development Kit to be used for running Java stored procedures and user-defined functions 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.

Related reference:

- "Maximum Java Interpreter Heap Size configuration parameter - java_heap_sz" on page 426

Federated Database System Support configuration parameter - 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 configuration parameter - sysadm_group" on page 521
- "System Control Authority Group Name configuration parameter - sysctrl_group" on page 522
- "System Maintenance Authority Group Name configuration parameter - sysmaint_group" on page 523
- "Authentication Type configuration parameter - authentication" on page 523
- "Use SNA Authentication configuration parameter - use_sna_auth" on page 525
- "Bypass Federated Authentication configuration parameter - fed_noauth" on page 525
- "Cataloging Allowed without Authority configuration parameter - catalog_noauth" on page 526

- "Default Database Path configuration parameter - dftdbpath" on page 526
- "Trust All Clients configuration parameter - trust_allclnts" on page 527
- "Trusted Clients Authentication configuration parameter - trust_clntauth" on page 528

System Administration Authority Group Name configuration parameter - sysadm_group

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

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 98 operating system the SYSADM group must be NULL.

  This parameter must be "NULL" for Windows 98 clients when system security is used because the 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.

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.

Related reference:

- "System Control Authority Group Name configuration parameter - sysctrl_group" on page 522
- "System Maintenance Authority Group Name configuration parameter - sysmaint_group" on page 523

System Control Authority Group Name configuration parameter - sysctrl_group

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

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 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the 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.

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.

Related reference:

- "System Administration Authority Group Name configuration parameter - sysadm_group" on page 521

- "System Maintenance Authority Group Name configuration parameter - sysmaint_group" on page 523

System Maintenance Authority Group Name configuration parameter - sysmaint_group

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

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 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the 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.

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.

Related reference:

- "System Administration Authority Group Name configuration parameter - sysadm_group" on page 521
- "System Control Authority Group Name configuration parameter - sysctrl_group" on page 522

Authentication Type configuration parameter - authentication

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] SERVER [CLIENT; SERVER; 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 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 and KRB_SERVER_ENCRYPT. These values provide the same function as SERVER 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.

Recommendation: Typically, the default (SERVER) is adequate.

Use SNA Authentication configuration parameter - use_sna_auth

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 online

Propagation Class Immediate

Default [Range] No [Yes; No]

When *use_sna_auth* is set to *yes* and *authentication* is set to *server*, inbound connections to the server that use the SNA protocol with security type SAME or PROGRAM are only authenticated at the SNA layer, and not by DB2.

Related reference:

- "Authentication Type configuration parameter - authentication" on page 523

Bypass Federated Authentication configuration parameter - fed_noauth

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 online

Propagation Class Immediate

Default [Range] No [Yes; No]

When *fed_noauth* is set to *yes*, *authentication* is set to *server* or *server_encrypt*, and *federated* is set to *yes*, then authentication at the instance is bypassed. It is assumed that authentication will happen at the data source. Exercise caution when *fed_noauth* is set to *yes*. Authentication is done at neither the client nor at DB2. Any user who knows the SYSADM authentication name can assume SYSADM authority for the federated server.

Related reference:

- "Authentication Type configuration parameter - authentication" on page 523
- "Federated Database System Support configuration parameter - federated" on page 520

Cataloging Allowed without Authority configuration parameter - 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

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

Database server with local and remote clients NO [NO (0) — YES (1)]

Client; 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 configuration parameter - dftdbpath

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

UNIX	Home directory of instance owner [any existing path]
Windows	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 a Windows 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 Windows, 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.

Trust All Clients configuration parameter - 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]

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 clients from DB2 for OS/390 and z/OS, 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.

Related reference:

- "Authentication Type configuration parameter - authentication" on page 523
- "Trusted Clients Authentication configuration parameter - trust_clntauth" on page 528

Trusted Clients Authentication configuration parameter - 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]

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 user ID 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.

Related reference:

- "Authentication Type configuration parameter - authentication" on page 523
- "Trust All Clients configuration parameter - trust_allclnts" on page 527

DB2 Administration Server

The following parameters related to the DB2 administration server:

- "DAS Discovery Mode configuration parameter - discover" on page 530
- "Name of the DB2 Server System configuration parameter - db2system" on page 530
- "DAS Administration Authority Group Name configuration parameter - dasadm_group" on page 531
- "Scheduler Mode configuration parameter - sched_enable" on page 532
- "Tools Catalog Database Instance configuration parameter - toolscat_inst" on page 532
- "Tools Catalog Database configuration parameter - toolscat_db" on page 533
- "Tools Catalog Database Schema configuration parameter - toolscat_schema" on page 533
- "SMTP Server configuration parameter - smtp_server" on page 534
- "Java Development Kit Installation Path DAS configuration parameter - jdk_path" on page 535
- "Execute Expired Tasks configuration parameter - exec_exp_task" on page 535
- "Scheduler User ID configuration parameter - sched_userid" on page 536
- "Location of Contact List configuration parameter - contact_host" on page 536
- "Authentication Type DAS configuration parameter - authentication" on page 537
- "DAS Code Page configuration parameter - das_codepage" on page 537
- "DAS Territory configuration parameter - das_territory" on page 538

DAS Discovery Mode configuration parameter - discover

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	SEARCH [DISABLE; KNOWN; SEARCH]

From an administration server perspective, this configuration parameter determines the type of discovery mode that is started when the DB2 Administration Server starts.

- If discover = SEARCH, the administration server handles 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, the administration server handles only KNOWN discovery requests from clients.
- If discover = DISABLE, then the administration server will not handle any type of discovery request. The information for this server system is essentially hidden from clients.

The default discovery mode is SEARCH.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "Name of the DB2 Server System configuration parameter - db2system" on page 530

Name of the DB2 Server System configuration parameter - db2system

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Default [Range]	TCP/IP host name [any valid system name]

This parameter 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 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, the setup program sets it equal to the computer name specified for the Windows system.
- On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.

Related reference:

- "DAS Discovery Mode configuration parameter - discover" on page 530

DAS Administration Authority Group Name configuration parameter - dasadm_group

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid group name]

DAS Administration (DASADM) authority is the highest level of authority within the DAS. This parameter defines the group name with DASADM authority for the DAS.

DASADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows NT and Windows 2000 operating systems, 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 DASADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the DASADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the DASADM group can be any valid UNIX group name.

This parameter can only be updated from a Version 8 CLP.

Scheduler Mode configuration parameter - sched_enable

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Off [On; Off]

This parameter indicates whether or not the Scheduler is started by the administration server. The Scheduler allows tools such as the Task Center to schedule and execute tasks at the administration server.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "Tools Catalog Database Instance configuration parameter - toolscat_inst" on page 532
- "Tools Catalog Database configuration parameter - toolscat_db" on page 533
- "Tools Catalog Database Schema configuration parameter - toolscat_schema" on page 533
- "SMTP Server configuration parameter - smtp_server" on page 534
- "Execute Expired Tasks configuration parameter - exec_exp_task" on page 535
- "Scheduler User ID configuration parameter - sched_userid" on page 536

Tools Catalog Database Instance configuration parameter - toolscat_inst

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid instance]

This parameter indicates the local instance that contains the tools catalog database used by the Scheduler. The tools catalog database may be local or remote to this instance. If the database is local, the instance must be configured for TCP/IP. If the database is remote, the node referenced by the database directory entry must be a TCP/IP node.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "Scheduler Mode configuration parameter - sched_enable" on page 532

- "Tools Catalog Database configuration parameter - toolscat_db" on page 533
- "Tools Catalog Database Schema configuration parameter - toolscat_schema" on page 533
- "SMTP Server configuration parameter - smtp_server" on page 534
- "Execute Expired Tasks configuration parameter - exec_exp_task" on page 535
- "Scheduler User ID configuration parameter - sched_userid" on page 536

Tools Catalog Database configuration parameter - toolscat_db

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid database alias]

This parameter indicates the tools catalog database used by the Scheduler. This database must be in the database directory of the instance specified by *toolscat_inst*.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "Scheduler Mode configuration parameter - sched_enable" on page 532
- "Tools Catalog Database Instance configuration parameter - toolscat_inst" on page 532
- "Tools Catalog Database Schema configuration parameter - toolscat_schema" on page 533
- "SMTP Server configuration parameter - smtp_server" on page 534
- "Execute Expired Tasks configuration parameter - exec_exp_task" on page 535
- "Scheduler User ID configuration parameter - sched_userid" on page 536

Tools Catalog Database Schema configuration parameter - toolscat_schema

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid schema]

This parameter indicates the schema of the tools catalog database used by the Scheduler. The schema is used to uniquely identify a set of tools catalog tables and views within the database.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "Scheduler Mode configuration parameter - sched_enable" on page 532
- "Tools Catalog Database Instance configuration parameter - toolscat_inst" on page 532
- "Tools Catalog Database configuration parameter - toolscat_db" on page 533
- "SMTP Server configuration parameter - smtp_server" on page 534
- "Execute Expired Tasks configuration parameter - exec_exp_task" on page 535
- "Scheduler User ID configuration parameter - sched_userid" on page 536

SMTP Server configuration parameter - smtp_server

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid SMTP server TCP/IP hostname]

When the Scheduler is on, this parameter indicates the SMTP server that the Scheduler will use to send e-mail and pager notifications.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "Scheduler Mode configuration parameter - sched_enable" on page 532
- "Tools Catalog Database Instance configuration parameter - toolscat_inst" on page 532
- "Tools Catalog Database configuration parameter - toolscat_db" on page 533
- "Tools Catalog Database Schema configuration parameter - toolscat_schema" on page 533
- "Execute Expired Tasks configuration parameter - exec_exp_task" on page 535
- "Scheduler User ID configuration parameter - sched_userid" on page 536

Java Development Kit Installation Path DAS configuration parameter - jdk_path

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid path]

This parameter specifies the directory under which the Java Development Kit to be used for running DB2 administration server functions is installed. 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.

This parameter can only be updated from a Version 8 CLP.

Execute Expired Tasks configuration parameter - exec_exp_task

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	No [Yes; No]

This parameter specifies whether or not the Scheduler will execute tasks that have been scheduled in the past, but have not yet been executed. The Scheduler only detects expired tasks when it starts up.

For example, if you have a job scheduled to run every Saturday, and the Scheduler is turned off on Friday and then restarted on Monday, the job scheduled for Saturday is now a job that is scheduled in the past. If *exec_exp_task* is set to Yes, your Saturday job will run when the Scheduler is restarted.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "Scheduler Mode configuration parameter - sched_enable" on page 532
- "Tools Catalog Database Instance configuration parameter - toolscat_inst" on page 532

- “Tools Catalog Database configuration parameter - toolscat_db” on page 533
- “Tools Catalog Database Schema configuration parameter - toolscat_schema” on page 533
- “SMTP Server configuration parameter - smtp_server” on page 534
- “Scheduler User ID configuration parameter - sched_userid” on page 536

Scheduler User ID configuration parameter - sched_userid

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Informational
Default [Range]	Null [any valid user ID]

This parameter specifies the user ID used by the Scheduler to connect to the tools catalog database. This parameter is only relevant if the tools catalog database is remote to the DB2 administration server.

The userid and password used by the Scheduler to connect to the remote tools catalog database are specified using the **db2admin** command.

Related reference:

- “Scheduler Mode configuration parameter - sched_enable” on page 532
- “Tools Catalog Database Instance configuration parameter - toolscat_inst” on page 532
- “Tools Catalog Database configuration parameter - toolscat_db” on page 533
- “Tools Catalog Database Schema configuration parameter - toolscat_schema” on page 533
- “SMTP Server configuration parameter - smtp_server” on page 534
- “Execute Expired Tasks configuration parameter - exec_exp_task” on page 535

Location of Contact List configuration parameter - contact_host

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid Version 8 DB2 administration server TCP/IP hostname]

This parameter specifies the location where the contact information used for notification by the Scheduler and the Health Monitor is stored. The location is defined to be a DB2 administration server's TCP/IP hostname. Allowing *contact_host* to be located on a remote DAS provides support for sharing a contact list across multiple DB2 administration servers. If *contact_host* is not specified, the DAS assumes the contact information is local.

This parameter can only be updated from a Version 8 CLP.

Authentication Type DAS configuration parameter - authentication

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT]

This parameter determines how and where authentication of a user takes place.

If authentication is SERVER_ENCRYPT, then the user ID and password are sent from the client to the server so authentication can take place on the server. Passwords sent over the network are encrypted.

A value of KERBEROS_ENCRYPT means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication.

Note: The KERBEROS_ENCRYPT authentication type is only supported on servers running Windows 2000.

This parameter can only be updated from a Version 8 CLP.

DAS Code Page configuration parameter - das_codepage

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid DB2 code page]

This parameter indicates the code page used by the DB2 administration server. If the parameter is null, then the default code page of the system is

used. This parameter should be compatible with the locale of the local DB2 instances. Otherwise, the DB2 administration server cannot communicate with the DB2 instances.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "DAS Territory configuration parameter - das_territory" on page 538

DAS Territory configuration parameter - das_territory

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid DB2 territory]

This parameter shows the territory used by the DB2 administration server. If the parameter is null, then the default territory of the system is used.

This parameter can only be updated from a Version 8 CLP.

Related reference:

- "DAS Code Page configuration parameter - das_codepage" on page 537

Part 4. Appendixes

Appendix A. DB2 Registry and Environment Variables

This chapter describes how to use the registry and enviroment variables and provides lists of the variables in each cateory with an explanation of their syntax and usage.

DB2 registry and environment variables

This section lists DB2® registry variables and environment variables that you may need to know about to get up and running. Each variable has a brief description. Some variables might not apply in your environment.

To view a list of all supported registry variables, execute the following command:

```
db2set -lr
```

To change the value for a variable in the current or default instance, execute the following command:

```
db2set registry_variable_name=new_value
```

Whether the DB2 environment variables DB2INSTANCE, DB2NODE, DB2PATH, and DB2INSTPROF are stored in the DB2 profile registries depends on your operating system. To update these environment variables, use the `set` command. These changes are in effect until the next time the system is rebooted. On UNIX® platforms, you can use the `export` command instead of the `set` command.

You must set the values for the changed registry variables before you execute the DB2START command.

Note: If a registry variable requires Boolean values as arguments, the values YES, 1, and ON are all equivalent and the the values NO, 0, and OFF are also equivalent. For any variable, you can specify any of the appropriate equivalent values.

Related concepts:

- "Environment Variables and the Profile Registry" in the *Administration Guide: Implementation*

Related tasks:

- "Setting DB2 registry variables at the user level in the LDAP environment" in the *Administration Guide: Implementation*

Related reference:

- "General registry variables" on page 542
- "System environment variables" on page 546
- "Communications variables" on page 548
- "Command-line variables" on page 553
- "MPP configuration variables" on page 554
- "SQL compiler variables" on page 556
- "Performance variables" on page 562
- "Data-links variables" on page 569
- "Miscellaneous variables" on page 571

Registry and environment variables by category

The following sections list the registry and environment variables according to the aspect of database manager or database behavior that they control.

General registry variables

Table 30. 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 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.		

Table 30. General Registry Variables (continued)

Variable Name	Operating System	Values
Description		
DB2DBMSADDR	Default= 0x20000000 for Windows NT 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 whether to disable closing the active log file when the on-line backup is completed. When an on-line backup completes, the last active log file is truncated, closed, and made available to be archived. This ensures that your on-line backup has a complete set of archived logs available for recovery. You might disable closing 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 large number of on-line backups each day, you might disable closing the last active log file. You might also disable closing 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. During the short interval between these two events that you may receive log full messages.		
DB2DISCOVERYTIME	Windows 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.		
DB2INSTDEF	Windows operating systems	Default=DB2

Table 30. General Registry Variables (continued)

Variable Name	Operating System	Values
Description		
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 32 767
The registry variable determines 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.		
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 if the DB2DBDFT environment variable has been defined with a default database.		
DB2LOCALE	All	Default: NO Values: YES or NO
Specifies whether the default ″C″ locale of a process is restored to the default ″C″ locale after calling DB2 and whether to restore the process locale back to the original 'C' after calling a DB2 function. If the original locale was not 'C', then this registry variable is ignored.		
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 log on each time Windows 3.x is started.		
DB2TERRITORY	All	Default: derived from the language ID, as specified by the operating system.

Table 30. General Registry Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies the region, or territory code of the client application, which influences date and time formats.		
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.		
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`). It is strongly recommended 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.) Troubleshooting information includes information about using the trace facility with clients.		
DB2TRCFLUSH	Windows 3.x and Macintosh	Default=NO Values: YES or NO
On Windows 3.x and Macintosh, DB2TRACEFLUSH can be used in conjunction with DB2TRACEON=`YES`. Setting DB2TRACEFLUSH=`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 DB2TRACEFLUSH=`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: 1-32 767
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

Table 30. General Registry Variables (continued)

Variable Name	Operating System	Values
Description		
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.		

Related concepts:

- "DB2 registry and environment variables" on page 541

System environment variables

Table 31. System Environment Variables

Variable Name	Operating System	Values
Description		
DB2CONNECT_IN_APP_PROCESS	All	Default=YES Values: YES or NO
When you set 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 can 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 (",").
Defines one or more Windows NT domains. Only users belonging to these domains will have their connection or attachment requests accepted. This registry variable should only be used under a pure Windows NT domain environment with DB2 servers and clients running DB2 Universal Database Version 7.1 (or later).		
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.		

Table 31. System Environment Variables (continued)

Variable Name	Operating System	Values
Description		
DB2INSTANCE	All	Default=DB2INSTDEF on 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	Windows operating systems	Default: null
The environment variable used to specify the location of the instance directory on Windows operating systems, if different than DB2PATH.		
DB2LIBPATH	UNIX	Default: null
Specifies the value of 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 variable. The **db2start** executable then reads the value of DB2LIBPATH and appends this value to the end of the DB2-constructed LIBPATH.		
DB2NODE	All	Default: null Values: 1 to 999
Used to specify the target logical node of a DB2 Enterprise Server 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 for each table space value that you specify. The degree of parallelism is determined by the prefetch size and extent size for the containers in the table space. For example, if the prefetch size is four times the extent size, then there are four extent-sized prefetch requests. The number of containers in the table space does not affect the number of prefetchers. To enable parallel I/O for all table spaces, use the wildcard character, "*". To enable parallel I/O for a subset of all table spaces, enter the list of table spaces. If there is more than one container, extent-size pieces of any full prefetch request are broken down into smaller requests executed in parallel based on the number of prefetchers. When this variable is not enabled, the number of prefetcher requests created is based on the number of containers in the table space.		
DB2PATH	Windows operating systems	Default: (varies by operating system)
The environment variable used to specify the directory where the product is installed on Windows 32-bit operating systems.		

Table 31. System Environment Variables (continued)

Variable Name	Operating System	Values
Description		
DB2_USE_PAGE_CONTAINER_TAG	All	Default: null Values: ON, null
By default, DB2 stores a container tag in the first extent of each DMS container, whether it is a file or a device. The container tag is the metadata for the container. Before DB2 Version 8.1, the container tag was stored in a single page, and it thus required less space in the container. To continue to store the container tag in a single page, set DB2_USE_PAGE_CONTAINER_TAG to ON. However, if you set this registry variable to ON when you use RAID devices for containers, I/O performance might degrade. Because for RAID devices you create table spaces with an extent size equal to or a multiple of the RAID stripe size, setting the DB2_USE_PAGE_CONTAINER_TAG to ON causes the extents not to line up with the RAID stripes. As a result, an I/O request might need to access more physical disks than would be optimal. Users are strongly advised against enabling this registry variable. To activate changes to this registry variable, issue a DB2STOP command and then enter a DB2START command.		

Related concepts:

- "DB2 registry and environment variables" on page 541

Communications variables

Table 32. Communications Variables

Variable Name	Operating System	Values
Description		
DB2CHECKCLIENTINTERVAL	AIX, server only	Default=0 Values: A numeric value greater than zero.
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.		

Table 32. Communications Variables (continued)

Variable Name	Operating System	Values
Description		
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 territory 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 probably not needed in such a situation because multi-threaded applications typically do not change their locale settings because it is not "thread-safe" to do so.		
DB2NBADAPTERS	Windows	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	Windows 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 ensure that the NetBIOS protocol checkup runs more often, freeing up memory and other system resources left when unexpected agent/session termination occurs.		
DB2NBINTRLISTENS	Windows server only	Default=1 Values: 1-10 Multiple values should be separated by commas

Table 32. Communications Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies the number of NetBIOS listen send commands (NCBs) that are asynchronously issued in readiness for remote client interrupts. This flexibility is provided for ″interrupt active″ environments to ensure that interrupt calls from remote clients can establish connections when servers are busy servicing other remote interrupts. Setting DB2NBINTRLISTENS to a lower value conserves 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	Windows 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	Windows server only	Default=10 Range: 1-99
Specifies the number of NetBIOS ″receive_any″ commands (NCBs) that the server issues and maintains during operation. This value is adjusted depending on the number of remote clients to which your server is connected. Lower values 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	Windows 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	Windows server only	Default=6 Range: 1-720
Specifies the number of send NetBIOS commands (NCBs) that the server reserves for use. This value can 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 might 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	Windows server only	Default=null Range: 5-254

Table 32. Communications Variables (continued)

Variable Name	Operating System	Values
Description		
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	Windows 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 is 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 backward compatibility only. It is strongly recommended that 0 not be used.		
DB2RETRY	Windows	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	Windows	Default=1 minute Range: 0-7 200 minutes
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	Windows, AIX, and Sun Solaris	Default=null

Table 32. Communications Variables (continued)

Variable Name	Operating System	Values
Description		
Used to solve the problem caused by: • More than one instance running on the same machine • A Version 6 or Version 7 instance running on the same machine attempting to register the same TP names. When the **db2start** command is invoked, the instance specified will start the APPC listeners for the following TP names: • DB2DRDA • x'07'6DB		
DB2SOSNDBUF	Windows NT	Default=32767
Specifies the value of TCP/IP send buffers on Windows NT operating systems.		
DB2SYSPLEX_SERVER	Windows NT, and UNIX	Default=null
Specifies whether SYSPLEX exploitation when connected to DB2 for OS/390 or z/OS 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-line processor **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
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 processes (on UNIX) or threads (on 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. **Note:** Having the number of connection managers set to one causes a drop in performance on remote connections in systems with a lot of users, frequent connects and disconnects, or both.		
DB2_VI_ENABLE	Windows NT	Default=OFF Values: ON or OFF

Table 32. Communications Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies whether to use the Virtual Interface (VI) Architecture 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 currently supported implementations all use the same library name.		
DB2_VI_DEVICE	Windows NT	Default=null Values: `nic0` or `VINIC`
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 NIC is allowed per Windows NT machine; multiple logical nodes on the same physical machine will share the same NIC. The symbolic device name "VINIC" must be in upper case and can only be used with Synfinity Interconnect. All other currently supported implementations use "nic0" as the symbolic device name.		

Related concepts:

- "DB2 registry and environment variables" on page 541

Command-line variables

Table 33. 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 sleeps before it checks whether the back-end process is active and establishes 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.		
DB2IQTIME	All	Default=5 seconds Minimum value: 1 second

Table 33. Command Line Variables (continued)

Variable Name	Operating System	Values
Description		
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.		

Related concepts:

- "DB2 registry and environment variables" on page 541

MPP configuration variables

Table 34. MPP Configuration Variables

Variable Name	Operating System	Values
DB2ATLD_PWFILE	DB2 UDB ESE 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 addresses password security concerns and allows the separation of AutoLoader configuration information from authentication information. This registry variable is no longer needed, but is retained for backward compatibility.		
DB2CHGPWD_ESE	DB2 UDB ESE on AIX and Windows NT	Default=null Values: YES or NO
Specifies whether you allow other users to change passwords on AIX or Windows NT ESE 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 being changed only 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. This variable is required only if you use the old *db2atld* utility instead of the new LOAD utility.		
DB2_FORCE_FCM_BP	AIX	Default=No Values: Yes or No

Table 34. MPP Configuration Variables (continued)

Variable Name	Operating System	Values
This registry variable is applicable to DB2 UDB ESE for AIX with multiple logical partitions. When DB2START is issued, DB2 allocates the FCM buffers either from the database global memory or from a separate shared memory segment, if there is not enough global memory available. These buffers are used by all FCM daemons for that instance on the same physical machine. The kind of memory allocated is largely dependent on the number of FCM buffers to be created, as specified by the *fcm_num_buffers* database manager configuration parameter. If the DB2_FORCE_FCM_BP variable is set to Yes, the FCM buffers are always created in a separate memory segment so that 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. Communicating through shared memory in is faster, but there is one fewer shared memory segment available for other uses, particularly for database buffer pools. Enabling the DB2_FORCE_FCM_BP registry variable thus reduces the maximum size of database buffer pools.		
DB2_NUM_FAILOVER_NODES	All	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.		
DB2_PARTITIONEDLOAD__DEFAULT	All supported ESE platforms	Default: YES; Range of values: YES / NO
The DB2_PARTITIONEDLOAD_DEFAULT registry variable lets users change the default behavior of the Load utility in an ESE environment when no ESE-specific Load options are specified. The default value is YES, which specifies that in an ESE environment if you do not specify ESE-specific Load options, loading is attempted on all partitions on which the target table is defined. When the value is NO, loading is attempted only on the partition to which the Load utility is currently connected.		
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.		

Related concepts:

- "DB2 registry and environment variables" on page 541

SQL compiler variables

Table 35. SQL Compiler Variables

Variable Name	Operating System	Values
Description		
DB2_ANTIJOIN	All	Default=NO in a ESE environment Default=YES in a non-ESE environment Values: YES or NO
For DB2 Universal Database ESE environments: when ″Yes″ is specified, the optimizer searches for opportunities to transform “NOT EXISTS” subqueries into anti-joins which can be processed more efficiently by DB2. For non-ESE environments: when ″No″ is specified, the optimizer limits the opportunities to transform “NOT EXISTS” subqueries into anti-joins.		
DB2_CORRELATED_PREDICATES	All	Default=Yes Values: Yes or No
The default for this variable is ″Yes″. When there are unique indexes on correlated columns in a join, and this registry variable is ″Yes″, the optimizer attempts to detect and compensate for correlation of join predicates. When this registry variable is ″Yes″, 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. Adjustment is also done for correlation of simple equality predicates like `WHERE C1=5 AND C2=10` if there is an index on C1 and C2. The index need not be unique but the equality predicate columns must cover all the columns in the index.		
DB2_HASH_JOIN	All	Default=YES Values: YES or NO
Specifies hash join as a possible join method when compiling an access plan.		
DB2_INLIST_TO_NLJN	All	Default=NO Values: YES or NO

Table 35. SQL Compiler Variables (continued)

Variable Name	Operating System	Values
Description		

In some situations, the SQL compiler can rewrite an IN list predicate to a join. For example, the following query:
 	
`SELECT *`
`FROM EMPLOYEE`
`WHERE DEPTNO IN ('D11', 'D21', 'E21')`
 	
could be written as:
`SELECT *`
`FROM EMPLOYEE, (VALUES 'D11', 'D21', 'E21') AS V(DNO)`
`WHERE DEPTNO = V.DNO`
 	
This revision might provide better performance if there is an index on DEPTNO. The list of values would be accessed first and joined to EMPLOYEE with a nested loop join using the index to apply the join predicate.
 	
Sometimes the optimizer does not have accurate information to determine the best join method for the rewritten version of the query. This can occur if the IN list contains parameter markers or host variables which prevent the optimizer from using catalog statistics to determine the selectivity. This registry variable causes the optimizer to favor nested loop joins to join the list of values, using the table that contributes the IN list as the inner table in the join.		

DB2_LIKE_VARCHAR	All	Default=Y,N Values: Y, N, S, or a floating point constant between 0 and 6.2

Table 35. SQL Compiler Variables (continued)

Variable Name	Operating System	Values
Description		

Controls the collection and use of sub-element statistics. These are statistics about the content of data in columns when the data has a structure in the form of a series of sub-fields or sub-elements delimited by blanks.
 	
This registry variable affects how the optimizer deals with a predicate of the form:
`COLUMN LIKE '%xxxxxx%'`
 	
where the xxxxxx is any string of characters.
 	
The syntax showing how this registry variable is used is:
`db2set DB2_LIKE_VARCHAR=[Y|N|S|num1] [,Y|N|S|num2]`
 	
where
• The term preceding the comma, or the only term to the right of the predicate, means the following but only for columns that do not have positive sub-element statistics:
– S – The optimizer estimates the length of each element in a series of elements concatenated together to form a column based on the length of the string enclosed in the % characters.
– Y – The default. Use a default value of 1.9 for the algorithm parameter. Use a variable-length sub-element algorithm with the algorithm parameter.
– N – Use a fixed-length sub-element algorithm.
– num1 – Use the value of num1 as the algorithm parameter with the variable length sub-element algorithm.
• The term following the comma means the following:
– N – The default. Do not collect or use sub-element statistics.
– Y – Collect sub-element statistics. Use a variable-length sub-element algorithm that uses the collected statistics together with the 1.9 default value for the algorithm parameter in the case of columns with positive sub-element statistics.
– num2 – Collect sub-element statistics. Use a variable-length sub-element algorithm that uses the collected statistics together with the value of num2 as the algorithm parameter in the case of columns with positive sub-element statistics.		

DB2_MINIMIZE_LIST_PREFETCH	All	Default=NO Values: YES or NO
List prefetch is a special table access method that involves retrieving the qualifying RIDs from the index, sorting them by page number and then prefetching the data pages. Sometimes the optimizer does not have accurate information to determine if list prefetch is a good access method. This might occur when predicate selectivities contain parameter markers or host variables that prevent the optimizer from using catalog statistics to determine the selectivity. This registry variable prevents the optimizer from considering list prefetch in such situations.		

Table 35. SQL Compiler Variables (continued)

Variable Name	Operating System	Values
Description		
DB2_SELECTIVITY	ALL	Default=No Values: Yes or No
This registry variable controls where the SELECTIVITY clause can be used in search conditions in SQL statements. When this registry variable is set to ″Yes″, the SELECTIVITY clause can be specified for the following predicates: • A basic predicate in which at least one expression contains host variables • A LIKE predicate in which the MATCH expression, predicate expression, or escape expression contains host variables		
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”. It 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 35. SQL Compiler Variables (continued)

Variable Name	Operating System	Values
Description		

Specifies whether the optimizer searches 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'))`		

DB2_REDUCED_OPTIMIZATION	All	Default=NO Values: NO, YES, Any integer, DISABLE

Table 35. SQL Compiler Variables (continued)

Variable Name	Operating System	Values
Description		
This registry variable lets you request either reduced optimization features or rigid use of optimization features at the specified optimization level. If you reduce the number of optimization techniques used, you also reduce time and resource use during optimization. **Note:** Although optimization time and resource use might be reduced, the risk of producing a less than optimal data access plan is increased. Use this registry variable only when advised by IBM or one of its partners. • If set to NO The optimizer does not change its optimization techniques. • If set to YES If the optimization level is 5 (the default) or lower, the optimizer disables some optimization techniques that might consume significant prepare time and resources but do not usually produce a better access plan. If the optimization level is exactly 5, the optimizer scales back or disables some additional techniques, which might further reduce optimization time and resource use, but also further increase the risk of a less than optimal access plan. For optimization levels lower than 5, some of these techniques might not be in effect in any case. If they are, however, they remain in effect. • If set to any integer The effect is the same as YES, with the following additional behavior for dynamically prepared queries optimized at level 5. If the total number of joins in any query block exceeds the setting, then the optimizer switches to greedy join enumeration instead of disabling additional optimization techniques as described above for level 5 optimization levels. which implies that the query will be optimized at a level similar to optimization level 2. • If set to DISABLE The behavior of the optimizer when unconstrained by this DB2_REDUCED_OPTIMIZATION variable is sometimes to dynamically reduce the optimization for dynamic queries at optimization level 5. This setting disables this behavior and requires the optimizer to perform full level 5 optimization. Note that the dynamic optimization reduction at optimization level 5 takes precedence over the behavior described for optimization level of exactly 5 when DB2_REDUCED_OPTIMIZATION is set to YES as well as the behavior described for the integer setting.		

Related concepts:

- “Optimization class guidelines” on page 88
- “Strategies for selecting optimal joins” on page 191
- “DB2 registry and environment variables” on page 541

Related reference:

- “Optimization classes” on page 90

Performance variables

Table 36. Performance Variables

Variable Name	Operating System	Values
Description		
DB2_APM_PERFORMANCE	All	Default=OFF Values: ON , OFF
Set this variable to ON to enable performance-related changes in the access plan manager (APM) that affect the behavior of the SQL cache (package cache). These settings are not usually recommended for production systems. They introduce some limitations, such as the possibility of out-of-package cache errors or increased memory use or both. Setting DB2_APM_PERFORMANCE to ON also enables the 'No Package Lock' mode. This mode allows the Global SQL Cache to operate without the use of package locks, which are internal system locks that protect cached package entries from being removed. The 'No Package Lock' mode might result in somewhat improved performance, but certain database operations are not allowed. These prohibited operations might include: operations that invalidate packages, operations that inoperate packages, and PRECOMPILE, BIND, and REBIND.		
DB2_AVOID_PREFETCH	All	Default=OFF, Values: ON or OFF
Specifies whether prefetch should be used during crash recovery. If DB2_AVOID_PREFETCH=ON, prefetch is not used.		
DB2_AWE	Windows 2000	Default=null Values: <entry>[,<entry>,...] where <entry>=<buffer pool ID>,<number of physical pages>, <number of address windows>
Allows DB2 UDB on 32-bit Windows 2000 platforms to allocate buffer pools that use up to 64 GB of memory. Windows 2000 must be configured correctly to support Address Windowing Extensions (AWE) buffer pools. This includes associating the "lock pages in memory"-right with the user, allocating the physical pages and the address window pages, and setting this registry variable. In setting this variable you need to know the buffer pool ID of the buffer pool that is to be used for AWE support. The ID of the buffer pool can be seen in the BUFFERPOOLID column of the SYSCAT.BUFFERPOOLS system catalog view. **Note:** If AWE support is enabled, extended storage cannot be used for any of the buffer pools in the database. Also, buffer pools referenced with this registry variable must already exist in SYSCAT.SYSBUFFERPOOLS.		
DB2_BINSORT	All	Default=YES Values: YES or NO

Table 36. 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 data types such as BIGINT, CHAR, VARCHAR, FLOAT, and DECIMAL, as well as combinations of these data types. To enable this new algorithm, use the following command: `db2set DB2_BINSORT = yes`		
DB2BPVARS	As specified for each parameter	Default=path

Two sets of parameters are available to tune buffer pools. One set of parameters, available only on Windows, specify that buffer pools should use scatter read for specific types of containers. The other set of parameters, available on all platforms, affect prefetching behavior.
 	
Parameters are specified in an ASCII file, one parameter on each line, in the form `parameter=value`. For example, a file named `bpvars.vars` might contain the following lines:
 	
`NO_NT_SCATTER = 1`
`NUMPREFETCHQUEUES = 2`
 	
Assuming that `bpvars.vars` is stored in `F:\vars\`, to set these variables you execute the following command:
`db2set DB2BPVARS=F:\vars\bpvars.vars`
 	
Scatter-read parameters
 	
The scatter-read parameters are recommended for systems with a large amount of sequential prefetching against the respective type of containers and for which you have already set DB2NTNOCACHE to ON. These parameters, available only on Windows platforms, are NT_SCATTER_DMSFILE, NT_SCATTER_DMSDEVICE, and NT_SCATTER_SMS. Specify the NO_NT_SCATTER parameter to explicitly disallow scatter read for any container. Specific parameters are used to turn scatter read on for all containers of the indicated type. For each of these parameters, the default is zero (or OFF); and the possible values include: zero (or OFF) and 1 (or ON).
Note: You can turn on scatter read only if DB2NTNOCACHE is set to ON to turn Windows file caching off. If DB2NTNOCACHE is set to OFF or not set, a warning message is written to the administration notification log if you attempt to turn on scatter read for any container, and scatter read remains disabled.		

Table 36. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
Prefetch-adjustment parameters The prefetch-adjustment parameters are NUMPREFETCHQUEUES and PREFETCHQUEUESIZE. These parameters are available on all platforms and can be used to improve buffer-pool data prefetching. For example, consider sequential prefetching in which the desired PREFETCHSIZE is divided into PREFETCHSIZE/EXTENTSIZE prefetch requests. In this case, requests are placed on prefetch queues from which I/O servers are dispatched to perform asynchronous I/O. By default, DB2 maintains one queue of size max(100 , 2*NUM_IOSERVERS) for each database partition. In some environments, performance improves with either more queues or queues of a different size or both. The number of prefetch queues should be at most one half of the number of I/O servers. When you set these parameters, consider other parameters such as PREFETCHSIZE, EXTENTSIZE, NUM_IOSERVERS, buffer-pool size, and DB2_BLOCK_BASED_BP, as well as workload characteristics such as the number of current users. If you think the default values are too small for your environment, first increase the values only slightly. For example, you might set NUMPREFETCHQUEUES=4 and PREFETCHQUEUESIZE=200. Make changes to these parameters in a controlled manner so that you can monitor and evaluate the effects of the change. For NUMPREFETCHQUEUES, the default is 1, and the range of values is 1 to NUM_IOSERVERS. If you set NUMPREFETCHQUEUES to less than 1, it is adjusted to 1. If you set it greater than NUM_IOSERVERS, it is adjusted to NUM_IOSERVERS. For PREFETCHQUEUESIZE, the default value is max(100,2*NUM_IOSERVERS). The range of values is 1 to 32767. If you set PREFETCHQUEUESIZE to less than 1, it is adjusted to the default. If set greater than 32767, it is adjusted to 32767.		
DB2CHKPTR	All	Default=OFF, Values: ON or OFF
Specifies whether or not pointer checking for input is required.		
DB2_ENABLE_BUFPD	All	Default=OFF, Values: ON or OFF
Specifies whether or not DB2 uses intermediate buffering to improve query performance. The buffering may not improve query performance in all environments. Testing should be done to determine individual query performance improvements.		
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.		

Table 36. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
DB2MAXFSCRSEARCH	All	Default=5 Values: -1, 1 to 33 554
Specifies the number of free-space control records to search when adding a record to a table. The default is to search five free-space control records. Modifying this value allows you to balance insert speed with space reuse. Use large values to optimize for space reuse. Use small values to optimize for insert speed. Setting the value to -1 forces the database manager to search all free-space control records.		
DB2MEMDISCLAIM	AIX	Default=YES Values: YES or NO
On AIX, memory used by DB2 processes may have some associated paging space. This paging space may remain reserved even when the associated memory has been freed. Whether or not this is so depends on the AIX system's (tunable) virtual memory management allocation policy. The DB2MEMDISCLAIM registry variable controls whether DB2 agents explicitly request that AIX disassociate the reserved paging space from the freed memory. A DB2MEMDISCLAIM setting of YES results in smaller paging space requirements, and possibly less disk activity from paging. A DB2MEMDISCLAIM setting of NO will result in larger paging space requirements, and possibly more disk activity from paging. In some situations, such as if paging space is plentiful and real memory is so plentiful that paging never occurs, a setting of NO provides a minor performance improvement.		
DB2MEMMAXFREE	All	Default= 8 388 608 bytes Values: 0 to 2^{32}-1 bytes
Specifies the maximum number of bytes of unused private memory, that is retained by DB2 processes before unused memory is returned to the operating system.		
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 caching by AIX of DB2 data read from JFS file systems into memory (outside the buffer pool). If you want 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_MMAP_WRITE	AIX	Default=ON Values: ON or OFF

Table 36. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
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 file systems 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.		
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 permits 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 DB2 opens 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. In Windows NT, files are cached when they are opened, which is is the default behavior. 1 MB is reserved from a system pool for every 1 GB in the file. Use this registry variable to override the undocumented 192 MB limit for the cache. When the cache limit is reached, an out-of-resource error is given.		
DB2NTPRICLASS	Windows NT	Default=null Value: R, H, (any other value)

Table 36. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
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, the working set of a process 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, overrides the current buffer pool size.		
DB2_PINNED_BP	AIX, HP-UX	Default=NO Values: YES or NO

Table 36. Performance Variables (continued)

Variable Name	Operating System	Values
Description		

This variable is used to specify the database global memory (including buffer pools) associated with the database in the main memory on some AIX operating systems. Keeping this database global memory in the system main memory allows database performance to be more consistent.
 	
For example, if the buffer pool is swapped out of the system main memory, database performance deteriorates. The reduction of disk I/O by having the buffer pools in system memory improves database performance. If other applications require more of the main memory, allow the database global memory to be swapped out of main memory, depending on the system main memory requirements.
 	
For HP-UX in a 64-bit environment, in addition to modifying this registry variable, the DB2 instance group must be given the MLOCK privilege. To do this, a user with root access rights performs the following actions:
1. Adds the DB2 instance group to the `/etc/privgroup` file. For example, if the DB2 instance group belongs to `db2iadm1` group then the following line must be added to the `/etc/privgroup` file:
`db2iadm1 MLOCK`
2. Issues the following command:
`setprivgrp -f /etc/privgroup`		

DB2PRIORITIES	All	Values setting is platform dependent.
Controls the priorities of DB2 processes and threads.		
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.		
DB2_STPROC_LOOKUP_FIRST	All	Default=OFF Values: ON or OFF

Table 36. Performance Variables (continued)

Variable Name	Operating System	Values
Description		
Formerly DB2_DARI_LOOKUP_ALL, this variable specifies whether or not the UDB server performs a catalog lookup for ALL DARIs and stored procedures before it looks 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 degrades performance because the catalog lookup might be performed on another node in an ESE configuration before the function directories are searched. When you call a stored procedure, the default behavior for DB2 is to search for a shared library with the same name as the stored procedure in the `function` subdirectory of the `sqllib` subdirectory and in the `unfenced` subdirectory of the `function` subdirectory of the `sqllib` subdirectory before it looks up the name of the shared library for the stored procedures in the system catalog. Only stored procedures of PARAMETER TYPE DB2DARI can have the same name as their shared library, so only DB2DARI stored procedures benefit from the default behavior of DB2. If you use stored procedures cataloged with a different PARAMETER TYPE, the time that DB2 spends searching the above directories degrades the performance of those stored procedures. To enhance the performance of stored procedures that are not cataloged as PARAMETER TYPE DB2DARI, set the DB2_STPROC_LOOKUP_FIRST registry variable to ON. This registry variable forces DB2 to look up the name of the shared library for the stored procedure in the system catalog before searching the above directories.		

Related concepts:

- "DB2 registry and environment variables" on page 541

Data-links variables

Table 37. Data Links Variables

Variable Name	Operating System	Values
Description		
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 placed 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 specified, it is used instead of the path specified with DLFM_BACKUP_DIR_NAME.		

Table 37. Data Links Variables (continued)

Variable Name	Operating System	Values
Description		
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
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 environment in which the server can take over files on its own machine. DFS indicates that the Data Links server is in a distributed file system (DFS) environment in which the server can take over files throughout the file system. Mixing DFS file sets and native file systems 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/opt/db2_08_01 /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

Table 37. Data Links Variables (continued)

Variable Name	Operating System	Values
Description		
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.		
DLFM_TSM_MGMTCLASS	AIX, Windows NT, Solaris	Default: the default TSM management class Values: any valid TSM management class
Specifies which TSM management class to use to archive and retrieve linked files. If no value is set for this variable, the default TSM management class is used.		

Related concepts:

- “DB2 registry and environment variables” on page 541

Miscellaneous variables

Table 38. Miscellaneous Variables

Variable Name	Operating System	Values
Description		
DB2ADMINSERVER	Windows and UNIX	Default=null
Specifies 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 before this option was 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.		

Table 38. Miscellaneous Variables (continued)

Variable Name	Operating System	Values
Description		
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 the 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
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_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.		

Table 38. Miscellaneous Variables (continued)

Variable Name	Operating System	Values
Description		
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.		

DB2LDAP_CLIENT_PROVIDER	Windows 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.		

Table 38. Miscellaneous Variables (continued)

Variable Name	Operating System	Values
Description		
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 to be rolled back. Any other setting results in transaction rollback.		
DB2_NEWLOGPATH2	UNIX	Default=0 Values: 0 or 1
This parameter allows you to specify whether a secondary path should be used to implement dual logging. The secondary path name is generated by appending a "2" to the current value of the *logpath* database configuration parameter.		
DB2NOEXITLIST	All	Default=OFF Values: ON or OFF
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 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.		
DB2SORCVBUF	Windows NT	Default=32767
Specifies the value of TCP/IP receive buffers on Windows NT operating systems.		
DB2SORT	All, server only	Default=null

Table 38. Miscellaneous Variables (continued)

Variable Name	Operating System	Values
Description		
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 and UNIX	Default=null
Specifies the name that is used by your users and database administrators to identify the DB2 server system. If possible, this name should be unique within your network. This name is displayed in the system level of the Control Center's object tree to aid administrators in the identification of server systems that can be administered from the Control Center. When using the 'Search the Network' function of the Client Configuration Assistant, DB2 discovery returns this name and it is displayed at the system level in the resulting object tree. This name aids users in identifying the system that contains the database they wish to access. A value for DB2SYSTEM is set at installation time as follows: • On Windows NT the setup program sets it equal to the computer name specified for the Windows system. • On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.		
DB2_VENDOR_INI	AIX, HP-UX, Sun Solaris, and Windows	Default=null Values: Any valid path and file.
Points to a file containing all vendor-specific environment settings. The value is read when the database manager starts.		
DB2_XBSA_LIBRARY	AIX, HP-UX, Sun Solaris, and Windows	Default=null Values: Any valid path and file.

Points to the vendor-supplied XBSA library. On AIX, the setting must include the shared object if it is not named shr.o. HP-UX, Sun Solaris, and Windows NT do not require the shared object name. For example, to use Legato's NetWorker Business Suite Module for DB2, the registry variable must be set as follows:
`db2set DB2_XSBA_LIBRARY="/usr/lib/libxdb2.a(bsashr10.o)"`
 	
The XBSA interface can be invoked through the BACKUP DATABASE or the RESTORE DATABASE commands. For example:
`db2 backup db sample use XBSA`
`db2 restore db sample use XBSA`		

Related concepts:

• "DB2 registry and environment variables" on page 541

Appendix B. Explain tables

Explain tables

The Explain tables capture access plans when the Explain facility is activated. The Explain tables must be created before Explain can be invoked. You can create them using the documented table definitions, or you can create them by invoking the sample command line processor (CLP) script provided in the EXPLAIN.DDL file located in the 'misc' subdirectory of the 'sqllib' directory. To invoke the script, connect to the database where the Explain tables are required, then issue the command:

```
db2 -tf EXPLAIN.DDL
```

The population of the Explain tables by the Explain facility will not activate triggers or 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.

Related reference:

- “EXPLAIN_ARGUMENT table” on page 578
- “EXPLAIN_OBJECT table” on page 585
- “EXPLAIN_OPERATOR table” on page 588
- “EXPLAIN_PREDICATE table” on page 590
- “EXPLAIN_STREAM table” on page 595
- “ADVISE_INDEX table” on page 597
- “ADVISE_WORKLOAD table” on page 600
- “EXPLAIN_INSTANCE table” on page 582
- “EXPLAIN_STATEMENT table” on page 592

EXPLAIN_ARGUMENT table

The EXPLAIN_ARGUMENT table represents the unique characteristics for each individual operator, if there are any.

Table 39. EXPLAIN_ARGUMENT Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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 40. 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.

Table 40. ARGUMENT_TYPE and ARGUMENT_VALUE column values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
DUPLWARN	TRUE FALSE	Duplicates Warning flag.
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 40. 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 40. 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.

Table 41. EXPLAIN_INSTANCE Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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
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

Table 41. EXPLAIN_INSTANCE Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
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.
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.

Table 41. EXPLAIN_INSTANCE Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
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 42. EXPLAIN_OBJECT Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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.
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.

Table 42. EXPLAIN_OBJECT Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
OVERHEAD	DOUBLE	No	No	Total estimated overhead, in milliseconds, for a single random I/O to the specified table space. Includes controller overhead, disk seek, and latency times. Set to -1 if no table space is involved.
TRANSFER_RATE	DOUBLE	No	No	Estimated time to read a data page, in milliseconds, from the specified table space. Set to -1 if no table space is involved.
PREFETCHSIZE	INTEGER	No	No	Number of data pages to be read when prefetch is performed. Set to -1 for a table function.
EXTENTSIZE	INTEGER	No	No	Size of extent, in data pages. This many pages are written to one container in the table space before switching to the next container. Set to -1 for a table function.
CLUSTER	DOUBLE	No	No	Degree of data clustering with the index. If >= 1, this is the CLUSTERRATIO. If >= 0 and < 1, this is the CLUSTERFACTOR. Set to -1 for a table, table function, or if this statistic is not available.
NLEAF	INTEGER	No	No	Number of leaf pages this index object's values occupy. Set to -1 for a table, table function, or if this statistic is not available.
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 43. 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 44. EXPLAIN_OPERATOR Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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.
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 instructions) of fetching the next row for the chosen access plan up to and including this operator.

Table 44. EXPLAIN_OPERATOR Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
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.
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 45. 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 46. EXPLAIN_PREDICATE Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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.
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.

Table 46. EXPLAIN_PREDICATE Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
RELOP_TYPE	CHAR(2)	No	No	The type of relational operator used in this predicate.
SUBQUERY	CHAR(1)	No	No	Whether or not a data stream from a subquery is required for this predicate. There may be multiple subquery streams required. Possible values are: **N** No subquery stream is required **Y** One or more subquery streams is required
FILTER_FACTOR	DOUBLE	No	No	The estimated fraction of rows that will be qualified by this predicate.
PREDICATE_TEXT	CLOB(1M)	Yes	No	The text of the predicate as recreated from the internal representation of the SQL statement. Null if not available.

Table 47. 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 48. 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.

Table 49. EXPLAIN_STATEMENT Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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 49. EXPLAIN_STATEMENT Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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 49. EXPLAIN_STATEMENT Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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 50. EXPLAIN_STREAM Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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
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.

Table 50. EXPLAIN_STREAM Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
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.
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 (HASHEDVALUE() or DBPARTITIONNUM()) **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 51. ADVISE_INDEX Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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.
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.

Table 51. ADVISE_INDEX Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
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.
SYSTEM_REQUIRED	SMALLINT	No	No	1 if one or the other of the following conditions is met: – This index is required for a primary or unique key constraint, or this index is a dimension block index or composite block index for a multi-dimensional clustering (MDC) table. – This is an index on the (OID) column of a typed table. 2 if both of the following conditions are met: – This index is required for a primary or unique key constraint, or this index is a dimension block index or composite block index for an MDC table. – This is an index on the (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.)

Table 51. ADVISE_INDEX Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
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)
MINPCTUSED	SMALLINT	No	No	If not zero, then online index defragmentation 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.

Table 52. ADVISE_WORKLOAD Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

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.

Appendix C. SQL explain tools

You can use the db2expln and dynexpln tools to understand the access plan chosen for a particular SQL statement. You can also use the integrated Explain Facility in the Control Center 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)
- The purpose and contents of the system catalog tables
- General application tuning concepts

The topics in this section provide information about `db2expln` and `dynexpln`.

SQL explain tools

The `db2expln` tool describes the access plan selected for SQL statements. It can be used to obtain a quick explanation of the chosen access plan when explain data was not captured. For static SQL, `db2expln` examines the packages stored in the system catalog tables. For dynamic SQL, `db2expln` examines the sections in the SQL cache.

The `dynexpln` tool can also be used to describe the access plan selected for dynamic statements. It creates a static package for the statements and then uses the `db2expln` tool to describe them. However, because the dynamic SQL can be examined by `db2expln` this utility is retained only for backward compatibility.

The explain tools (`db2expln` and `dynexpln`) are located in the `bin` 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 uses the db2expln.bnd, db2exsrv.bnd, and db2exdyn.bnd files to bind itself to a database the first time the database is accessed.

To run db2expln, you must have the SELECT privilege on the system catalog views as well as the EXECUTE privilege for the db2expln, db2exsrv, and db2exdyn packages. To run dynexpln, you must have BINDADD authority for the database, and the schema you are using to connect to the database must exist or you must have the IMPLICIT_SCHEMA authority for the database. To explain dynamic SQL using either db2expln or dynexpln, you must also have 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.)

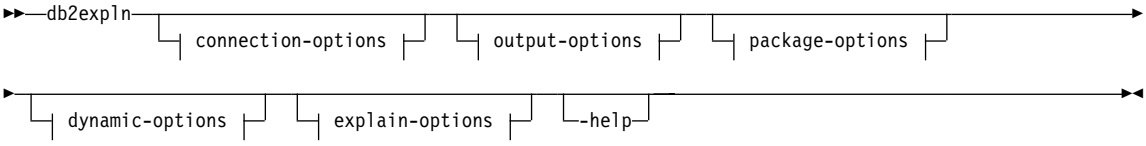
Related concepts:

- "SQL explain facility" on page 227
- "Guidelines for capturing explain information" on page 239
- "Guidelines for analyzing explain information" on page 241

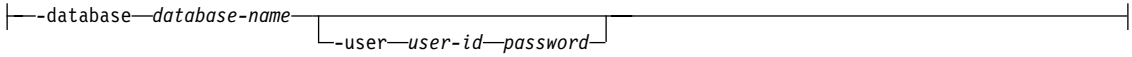
db2expln

The following sections describe the syntax and parameters for *db2expln* and provide usage notes.

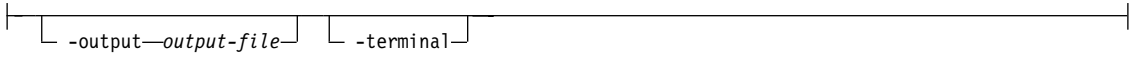
db2expln syntax and parameters



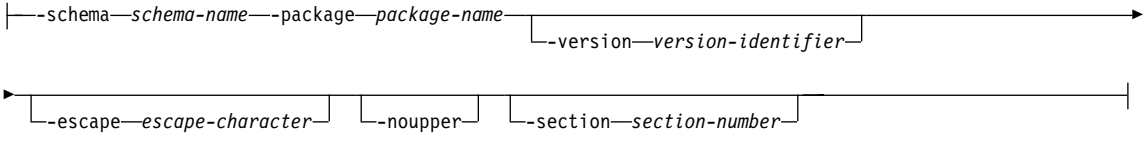
connection-options:



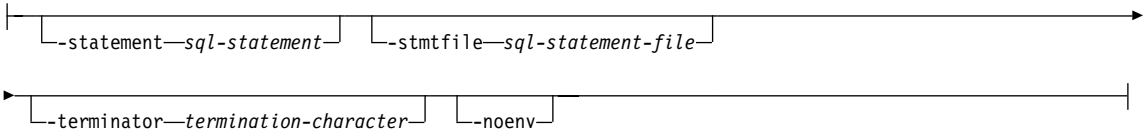
output-options:



package-options:



dynamic-options:



explain-options:

The options may be specified in any order.

connection-options:

These options specify the database to connect to and any options necessary to make the connection. The connection options are required except when the **-help** option is specified.

-database *database-name*
The name of the database that contains the packages to be explained.

For backward compatibility, you can use **-d** instead of **-database**.

-user *user-id password*
The authorization ID and password to use when establishing the database connection. Both *user-id* and *password* must be valid according to DB2® naming conventions and must be recognized by the database.

For backward compatibility, you can use **-u** instead of **-user**.

output-options:

These options specify where the `db2expln` output should be directed. Except when the **-help** option is specified, you must specify at least one output option. If you specify both options, output is sent to a file as well as to the terminal.

-output *output-file*
The output of `db2expln` is written to the file that you specify.

For backward compatibility, you can use **-o** instead of **-output**.

-terminal
The `db2expln` output is directed to the terminal.

For backward compatibility, you can use **-t** instead of **-terminal**.

package-options:

These options specify one or more packages and sections to be explained. Only static SQL in the packages and sections is explained.

Note: As in a LIKE predicate, you can use the pattern matching characters, which are percent sign (%) and underscore (_), to specify the *schema-name*, *package-name*, and *version-identifier*.

-schema *schema-name*
The schema of the package or packages to be explained.

For backward compatibility, you can use **-c** instead of **-schema**.

-package *package-name*
The name of the package or packages to be explained.

For backward compatibility, you can use **-p** instead of **-schema**.

-version *version-identifier*
The version identifier of the package or packages to be explained. The default version is the empty string.

-escape *escape-character*

The character, *escape-character* to be used as the escape character for pattern matching in the *schema-name*, *package-name*, and *version-identifier*.

For example, the `db2expln` command to explain the package TESTID.CALC% is as follows:

```
db2expln -schema TESTID -package CALC% ....
```

However, this command would also explain any other plans that start with CALC. To explain only the TESTID.CALC% package, you must use an escape character. If you specify the exclamation point (!) as the escape character, you can change the command to read: `db2expln -schema TESTID -escape ! -package CALC!% ...` . Then the ! character is used as an escape character and thus !% is interpreted as the % character and not as the ″match anything″ pattern. There is no default escape character.

For backward compatibility, you can use **-e** instead of **-escape**.

Note: To avoid problems, do not specify the operating system escape character as the `db2expln` escape character.

-noupper

Specifies that the *schema-name*, *package-name*, and *version-identifier*, should not be converted to upper case before searching for matching packages.

By default, these variables are converted to upper case before searching for packages. This option indicates that these values should be used exactly as typed.

For backward compatibility, you can use **-l**, which is a lowercase L and not the number 1, instead of **-noupper**.

-section *section-number*

The section number to explain within the selected package or packages.

To explain all the sections in each package, use the number zero (0). This is the default behavior. If you do not specify this option, or if *schema-name*, *package-name*, or *version-identifier* contain a pattern-matching character, all sections are displayed.

To find section numbers, query the system catalog view SYSCAT.STATEMENTS. Refer to the *SQL Reference* for a description of the system catalog views.

For backward compatibility, you can use **-s** instead of **-section**.

dynamic-options:

These options specify one or more dynamic SQL statements to be explained.

-statement *sql-statement*
An SQL statement to be dynamically prepared and explained. To explain more than one statement, either use the **-stmtfile** option to provide a file containing the SQL statements to explain, or use the **-terminator** option to define a termination character that can be used to separate statements in the **-statement** option.

For compatibility with `dynexpln`, you can use **-q** instead of **-statement**.

-stmtfile *sql-statement-file*
A file that contains one or more SQL statements to be dynamically prepared and explained. By default, each line of the file is assumed to be a distinct SQL statement. If statements must span lines, use the **-terminator** option to specify the character that marks the end of an SQL statement.

For compatibility with `dynexpln`, you can use **-f** instead of **-stmtfile**.

-terminator *termination-character*
The character that indicates the end of dynamic SQL statements. By default, the **-statement** option provides a single SQL statement and each line of the file in the **-stmtfile** is treated as a separate SQL statement. The termination character that you specify can be used to provide multiple SQL statements with **-statement** or to have statements span lines in the **-stmtfile** file.

For compatibility with `dynexpln`, you can use **-z** instead of **-terminator**.

-noenv
Specifies that dynamic statements that alter the compilation environment should not be executed after they have been explained.

By default, db2expln will execute any of the following statements after they have been explained:

```
SET CURRENT DEFAULT TRANSFORM GROUP
SET CURRENT DEGREE
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
SET CURRENT QUERY OPTIMIZATION
SET CURRENT REFRESH AGE
SET PATH
SET SCHEMA
```

These statements make it possible to alter the plan chosen for subsequent dynamic SQL statements processed by `db2expln`.

If you specify **-noenv**, then these statement are explained, but not executed.

It is necessary to specify either **-statement** or **-stmtfile** to explain dynamic SQL. Both options may be specified in a single invocation of `db2expln`.

explain-options:

These options determine what additional information is provided in the explained plans.

-graph Show optimizer plan graphs. Each section is examined, and the original optimizer plan graph is constructed as presented by Visual Explain. Note that the generated graph may not match the Visual Explain graph exactly.

For backward compatibility, you can specify **-g** instead of **-graph**.

-opids Display operator ID numbers in the explained plan.

The operator ID numbers allow the output from `db2expln` to be matched to the output from the explain facility. Note that not all operators have an ID number and that some ID numbers that appear in the explain facility output do not appear in the db2expln output.

For backward compatibility, you can specify **-i** instead of **-opids**.

-help Shows the help text for `db2expln`. If this option is specified no packages are explained.

Most of the command line is processed in the *db2exsrv* stored procedure. To get help on all the available options, it is necessary to provide **connection-options** along with **-help**. For example, use:

```
db2expln -help -database SAMPLE
```

For backward compatibility, you can specify **-h** or **-?**.

Unless you specify the **-help** option, you must specify either package-options or dynamic-options. You can explain both packages and dynamic SQL with a single invocation of db2expln.

Some of the option flags above might have special meaning to your operating system and, as a result, might not be interpreted correctly in the `db2expln` command line. However, you might be able to enter these characters by preceding them with an operating system escape character. For more information, see your operating system documentation. Make sure that you do not inadvertently specify the operating system escape character as the `db2expln` escape character.

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.

To explain multiple plans with one invocation of db2expln, use the **-package**, **-schema**, and **-version** option and specify 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 -database SAMPLE -schema % -package %  -output my.exp
```

As another example, suppose a user has a CLP script file called ″statements.db2″ and wants to explain the statements in the file. The file contains the following statements:

```
SET PATH=SYSIBM, SYSFUN, DEPT01, DEPT93@
SELECT EMPNO, TITLE(JOBID) FROM EMPLOYEE@
```

To explain these statements, enter the following command:

```
db2expln -database DEPTDATA -stmtfile statements.db2 -terminator @ -terminal
```

Related concepts:

- “SQL explain tools” on page 601
- “Description of db2expln and dynexpln output” on page 610
- “Examples of db2expln and dynexpln output” on page 633

Usage notes for db2expln

The following are common messages displayed by db2expln:

- `No packages found for database package pattern: "<creator>".<package> with version "<version>"`

  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 (all sections) 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.

- Bind messages for <bind file> can be found in <message file>

  This message will appear if the bind of the given bind file was not successful. Further information on the problems encountered will be found in the given message file on the database server.

- `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.

- `Package "<creator>"."<package>", "<version>", is not valid. Rebind the package 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.

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

The dynexpln tool is still available for backward compatibility. However, you can use the **dynamic-options** of db2expln to perform all of the functions of dynexpln.

When you use the dynamic-options of db2expln, the statement is prepared as true dynamic SQL and the generated plan is explained from the SQL cache. This explain-output method provides more accurate access plans than dynexpln, which prepares the statement as static SQL. It also allows the use of features available only in dynamic SQL, such as parameter markers.

Related concepts:

- "SQL explain tools" on page 601
- "Examples of db2expln and dynexpln output" on page 633

Explain output information

The following sections describe the kind of information that *db2expln* and *dynexpln* can provide.

Description of db2expln and dynexpln output

In the output, the explain information for each package appears in the following 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. Below the section information, the explain output of the access plan chosen for the SQL statement appears.

The steps of an access plan, or section, are presented in the order that the database manager executes them. Each major step is shown as a left-justified heading with information about that step indented below it. Indentation bars appear in the left margin of the explain output for the access plan. These bars also mark the scope of the operation. Operations at a lower level of indentation, farther to the right, in the same operation are processed before returning to the previous level of indentation.

Remember that the access plan chosen was based on an augmented version of the original SQL statement that is shown in the output. For example, the original statement may cause triggers and constraints to be activated. In addition, the query rewrite component of the SQL compiler might rewrite the SQL statement to an equivalent but more efficient format. All of these factors are included in the information that the optimizer uses 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 you might expect for the original SQL statement. The SQL Explain facility, which includes the explain tables, SET CURRENT EXPLAIN mode, and Visual Explain, 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 you compare output from `db2expln` or `dynexpln` to the output of the Explain facility, the operator ID option (**-opids**) can be very useful. Each time `db2expln` or `dynexpln` starts processing a new operator from the Explain facility, the operator ID number is 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`.

Related concepts:

- "db2expln syntax and parameters" on page 602
- "dynexpln" on page 610
- "Table access information" on page 611
- "Temporary table information" on page 617
- "Join information" on page 620
- "Data stream information" on page 622
- "Insert, update, and delete information" on page 623
- "Block and row identifier preparation information" on page 624
- "Aggregation information" on page 625
- "Parallel processing information" on page 626
- "Federated query information" on page 629
- "Miscellaneous information" on page 630

Table access information

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 Materialized Query Table Name:

  ```
  Access Materialized Query 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
- Block access
- Parallel scan
- Scan directive
- Row access method
- Lock intents
- Predicates
- Miscellaneous statements

Number of Columns

The following statement indicates the number of columns being used from each row of the table:

```
#Columns = n
```

Block Access

The following statement indicates that the table has one or more dimension block indexes defined on it:

```
Clustered by Dimension for Block Index Access
```

If this text is not shown, the table was created without the DIMENSION clause.

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 type
| 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.
- Index type is one of:

```
Regular Index (Not Clustered)
Regular Index (Clustered)
Dimension Block Index
Composite Dimension Block Index
```

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, only 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
  ```

– If the qualifying rows are being accessed by using row IDs (RIDs) that were prepared earlier in the access plan, it will be indicated with the statement:

```
Fetch Direct Using Row IDs
```

If the table has one or more block indexes defined for it, then rows may be accessed by either block or row IDs. This is indicated by:

```
Fetch Direct Using Block or Row IOs
```

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

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 for each block of data retrieved from a blocked index.

   ```
   Block Predicates(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
   ```

3. The following statement indicates the number of predicates that will be evaluated once the data has been returned:

   ```
   Residual 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 statement:

  ```
  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
  ```

Related concepts:

- “Description of db2expln and dynexpln output” on page 610
- “Examples of db2expln and dynexpln output” on page 633

Temporary table information

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
  ```

- 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.

- 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. Table functions are indicated by:

```
Access User Defined Table Function
|  Name = schema.funcname
|  Specific Name = specificname
|  SQL Access Level = accesslevel
|  Language = lang
|  Parameter Style = parmstyle
|  Fenced                      Not Deterministic
|  Called on NULL Input        Disallow Parallel
|  Not Federated               Not Threadsafe
```

The specific name uniquely identifies the table function invoked. The remaining rows detail the atributes of the function.

Related concepts:

- “Description of db2expln and dynexpln output” on page 610
- “Examples of db2expln and dynexpln output” on page 633

Join information

There are three types of 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
Merge Join
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
Left Outer Merge Join
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 performance 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.

Related concepts:

- "Description of db2expln and dynexpln output" on page 610
- "Examples of db2expln and dynexpln output" on page 633

Data stream information

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
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 610
- "Examples of db2expln and dynexpln output" on page 633

Insert, update, and delete information

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: Materialized Query Table = schema.name ID = ts,n
Update: Materialized Query Table = schema.name ID = ts,n
Delete: Materialized Query Table = 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
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 610
- "Examples of db2expln and dynexpln output" on page 633

Block and row identifier preparation information

For some access plans, it is more efficient if the qualifying row and block identifiers (IDs) are sorted and duplicates removed (in the case of index ORing) or that a technique is used to identify IDs 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 ID preparation as indicated by the explain statements:

- Either of the following statements indicates that Index ORing is used to prepare the list of qualifying IDs:

  ```
  Index ORing Preparation
  Block Index ORing Preparation
  ```

  Index ORing refers to the technique of making more than one index access and combining the results to include the distinct IDs 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 ID preparation is to prepare the input data to be used during list prefetch, as indicated by either of the following:

  ```
  List Prefetch Preparation
  Block List Prefetch RID Preparation
  ```
- *Index ANDing* refers to the technique of making more than one index access and combining the results to include IDs that appear in all of the indexes accessed. Index ANDing processing is started with either of these statements:

  ```
  Index ANDing
  Block 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 IDs and use bit filter techniques to determine the IDs which appear in every index accessed. The following statements indicate that IDs are being processed for index ANDing:

```
Index ANDing Bitmap Build Using Row IDs
Index ANDing Bitmap Probe Using Row IDs
Index ANDing Bitmap Build and Probe Using Row IDs
Block Index ANDing Bitmap Build Using Block IDs
Block Index ANDing Bitmap Build and Probe Using Block IDs
Block Index ANDing Bitmap Build and Probe Using Row IDs
Block Index ANDing Bitmap Probe Using Block IDs and Build Using Row IDs
Block Index ANDing Bitmap Probe Using Block IDs
Block Index ANDing Bitmap Probe Using Row IDs
```

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 ID preparation, if list prefetch can be performed it will be indicated with the statement:

```
Prefetch: Enabled
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 610
- "Examples of db2expln and dynexpln output" on page 633

Aggregation information

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
```

Related concepts:

- “Description of db2expln and dynexpln output” on page 610
- “Examples of db2expln and dynexpln output” on page 633

Parallel processing information

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 database partition group 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 database partition group. (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
  ```

- Either of the following indicates that the subsection will be executed on the coordinator node.

  ```
  Directed to Application Coordinator Node
  Directed to Local 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:

    All rows are sent to the coordinator node:

    ```
    Broadcast to Coordinator Node
    ```

    All rows are sent to every database partition where the given subsection is running:

    ```
    Broadcast to All Nodes of Subsection n
    ```

    Each row is sent to a database partition based on the values in the row:

```
Hash to Specific Node
```

Each row is sent to a database partition that is determined while the statement is executing:

```
Send to Specific Node
```

Each row is sent to a randomly determined node:

```
Send to Random Node
```

- 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
  ```

Related concepts:

- "Description of db2expln and dynexpln output" on page 610
- "Examples of db2expln and dynexpln output" on page 633

Federated query information

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 read:

```
Ship 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
```

An insert, update, or delete operation that occurs at a data source will be indicated by the appropriate message:

```
Ship Distributed Insert #n
Ship Distributed Update #n
Ship Distributed Delete #n
```

If a table is being explicitly locked at a data source, this will be indicated with the statement:

```
Ship Distributed Lock Table #n
```

DDL statements against a data source are split into two parts. The part invoked at the data source is indicated by:

```
Ship Distributed DDL Statement #n
```

If the federated server is a partitioned database, then part of the DDL statement must be run at he catalog node. This is indicated by:

```
Distributed DDL Statement #n Completion
```

The detail for each distributed substatement is provided separately. The options for distributed statements 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
```

- If the data source is relational, the SQL for the substatement is displayed as:

```
SQL Statement:
     statement
```

  Non-relational data sources are indicated with:

```
Non-Relational Data Source
```

- The nicknames referenced in the substatement are listed as follows:

```
Nicknames Referenced:
   schema.nickname ID = n
```

  If the data source is relational, the base table for the nickname is shows as:

```
Base = baseschema.basetable
```

  If the data source is non-relational, the source file for the nickname is shown as:

```
Source File = filename
```

- If values are passed from the federated server to the data source before executing the substatement, 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 substatement, the number of values will be shown by:

```
#Output Columns: n
```

Related concepts:

- "Guidelines for analyzing where a federated query is evaluated" on page 218
- "Description of db2expln and dynexpln output" on page 610

Miscellaneous information

- 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
  ```

  If the position operation is against a federated data source, then the statement is:

  ```
  Distributed 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
  ```

- The following information will appear if a stored procedure is being invoked:

  ```
  Call Stored Procedure
  |   Name = schema.funcname
  |   Specific Name = specificname
  |   SQL Access Level = accesslevel
  |   Language = lang
  |   Parameter Style = parmstyle
  |   Expected Result Sets = n
  |   Fenced                      Not Deterministic
  |   Called on NULL Input        Disallow Parallel
  |   Not Federated               Not Threadsafe
  ```

- The following information will appear if one or more LOB locators are being freed:

  ```
  Free LOB Locators
  ```

Related concepts:

- “Description of db2expln and dynexpln output” on page 610
- “Examples of db2expln and dynexpln output” on page 633

Examples of db2expln and dynexpln Output

The following output examples show the explain information collected for specific queries in specific environments.

Examples of db2expln and dynexpln output

Five examples shown here can help you understand the layout and format of the output from `db2expln` and `dynexpln`. These examples were run against the SAMPLE database that is provided with DB2. A brief discussion is included with each example. Significant differences from one example to the next have been shown in **bold**.

Related concepts:

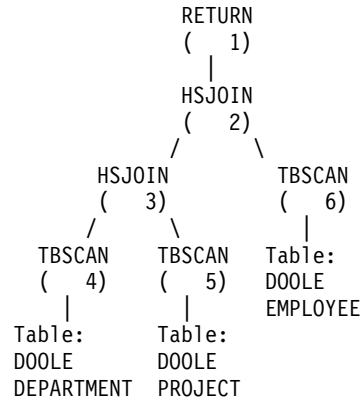
- "db2expln syntax and parameters" on page 602
- "dynexpln" on page 610
- "Example one: no parallelism" on page 633
- "Example two: single-partition plan with intra-partition parallelism" on page 635
- "Example three: multipartition plan with inter-partition parallelism" on page 637
- "Example four: multipartition plan with inter-partition and intra-partition parallelism" on page 640
- "Example five: federated database plan" on page 642

Example one: no parallelism

This example is simply requesting a list of all employee names, their jobs, department name and location, and the project names on which they are working. The essence of this access plan is that hash 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 as it is joined.

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

Prep Date = 2002/01/04
Prep Time = 14:05:00

Bind Timestamp = 2002-01-04-14.05.00.415403

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = No
```

```
SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION --------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 120.518692
Estimated Cardinality = 221.535980

(    6) Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
         |  #Columns = 3
         |  Relation Scan
         |  |  Prefetch: Eligible
         |  Lock Intents
         |  |  Table: Intent Share
         |  |  Row  : Next Key Share
(    6) |  Process Build Table for Hash Join
(    2) Hash Join
         |  Estimated Build Size: 7111
         |  Estimated Probe Size: 9457
(    5)  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
         |  |  #Columns = 2
         |  |  Relation Scan
         |  |  |  Prefetch: Eligible
         |  |  Lock Intents
         |  |  |  Table: Intent Share
         |  |  |  Row  : Next Key Share
(    5)  |  |  Process Build Table for Hash Join
(    3)  |  Hash Join
         |  |  Estimated Build Size: 5737
         |  |  Estimated Probe Size: 6421
(    4)  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
         |  |  |  #Columns = 3
         |  |  |  Relation Scan
         |  |  |  |  Prefetch: Eligible
         |  |  |  Lock Intents
         |  |  |  |  Table: Intent Share
         |  |  |  |  Row  : Next Key Share
(    4)  |  |  |  Process Probe Table for Hash Join
(    1) Return Data to Application
         |  #Columns = 5

End of section


Optimizer Plan:
```



The first part of the plan accesses the DEPARTMENT and PROJECT tables and uses a hash 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 plan with intra-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled for a four-way SMP machine.

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

Prep Date = 2002/01/04
Prep Time = 14:12:38

Bind Timestamp = 2002-01-04-14.12.38.732627

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = Yes (Bind Degree = 4)

SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ----------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno
```

```
Intra-Partition Parallelism Degree = 4

Estimated Cost        = 133.934692
Estimated Cardinality = 221.535980

(    2) Process Using 4 Subagents
(    7)   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
(    7)     Process Build Table for Hash Join
(    3)   Hash Join
            Estimated Build Size: 7111
            Estimated Probe Size: 9457
(    6)     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
(    6)       Process Build Table for Hash Join
(    4)     Hash Join
              Estimated Build Size: 5737
              Estimated Probe Size: 6421
(    5)       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
(    5)         Process Probe Table for Hash Join
(    2)   Insert Into Asynchronous Local Table Queue  ID = q1
(    2) Access Local Table Queue  ID = q1  #Columns = 5
(    1) Return Data to Application
          #Columns = 5

End of section


Optimizer Plan:

                RETURN
                (   1)
                  |
                 LTQ
                (   2)
                  |
```

```
                HSJOIN
                (   3)
               /      \
         HSJOIN        TBSCAN
         (   4)        (   7)
        /      \         |
    TBSCAN    TBSCAN   Table:
    (   5)    (   6)   DOOLE
      |         |      EMPLOYEE
  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.

Example three: multipartition plan with inter-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled on a partitioned database made up of three database partitions.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

Prep Date = 2002/01/04
Prep Time = 14:54:57

Bind Timestamp = 2002-01-04-14.54.57.033666

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = Yes
Intra-Partition Parallel = No

SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno
```

```
Estimated Cost        = 118.483406
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)     Distribute Subsection #2
            |  Broadcast to Node List
            |  |  Nodes = 10, 33, 55
(-----)     Distribute Subsection #3
            |  Broadcast to Node List
            |  |  Nodes = 10, 33, 55
(-----)     Distribute Subsection #1
            |  Broadcast to Node List
            |  |  Nodes = 10, 33, 55
(    2)     Access Table Queue  ID = q1  #Columns = 5
(    1)     Return Data to Application
            |  #Columns = 5

        Subsection #1:
(    8)     Access Table Queue  ID = q2  #Columns = 2
(    3)     Hash Join
            |  Estimated Build Size: 5737
            |  Estimated Probe Size: 8015
(    6)     |  Access Table Queue  ID = q3  #Columns = 3
(    4)     |  Hash Join
            |  |  Estimated Build Size: 5333
            |  |  Estimated Probe Size: 6421
(    5)     |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
            |  |  |  #Columns = 3
            |  |  |  Relation Scan
            |  |  |  |  Prefetch: Eligible
            |  |  |  Lock Intents
            |  |  |  |  Table: Intent Share
            |  |  |  |  Row  : Next Key Share
(    5)     |  |  |  Process Probe Table for Hash Join
(    2)     Insert Into Asynchronous Table Queue  ID = q1
            |  Broadcast to Coordinator Node
            |  Rows Can Overflow to Temporary Table

        Subsection #2:
(    9)     Access Table Name = DOOLE.PROJECT  ID = 2,7
            |  #Columns = 2
            |  Relation Scan
            |  |  Prefetch: Eligible
            |  Lock Intents
            |  |  Table: Intent Share
            |  |  Row  : Next Key Share
(    9)     |  Insert Into Asynchronous Table Queue  ID = q2
            |  |  Hash to Specific Node
            |  |  Rows Can Overflow to Temporary Tables
(    8)     Insert Into Asynchronous Table Queue Completion  ID = q2

        Subsection #3:
(    7)     Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
            |  #Columns = 3
            |  Relation Scan
```

```
            |  |  Prefetch: Eligible
            |  Lock Intents
            |  |  Table: Intent Share
            |  |  Row  : Next Key Share
(    7)     |  Insert Into Asynchronous Table Queue  ID = q3
            |  |  Hash to Specific Node
            |  |  Rows Can Overflow to Temporary Tables
(    6)    Insert Into Asynchronous Table Queue Completion  ID = q3

End of section


Optimizer Plan:

                RETURN
                (   1)
                  |
                 BTQ
                (   2)
                  |
                HSJOIN
                (   3)
               /      \
         HSJOIN        DTQ
         (   4)       (   8)
        /      \        |
    TBSCAN      DTQ     TBSCAN
    (   5)     (   6)   (   9)
      |          |       |
 Table:       TBSCAN    Table:
 DOOLE        (   7)    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 hash join to join it with the data from table queue q3. A second hash join then adds in the data from the DEPARTMENT table. The joined rows are then sent to the coordinator subsection using table queue q1.
- **Subsection #2**. This subsection scans the PROJECT table and hashes to a specific node with the results. These results are read by Subsection #1.
- **Subsection #3**. This subsection scans the EMPLOYEE table and hashes to a specific node with the results. These results are read by Subsection #1.

Example four: multipartition plan with inter-partition and intra-partition parallelism

This example shows the same SQL statement as the first example, 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"."EXAMPLE"  Version = ""

Prep Date = 2002/01/04
Prep Time = 14:58:35

Bind Timestamp = 2002-01-04-14.58.35.169555

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = Yes
Intra-Partition Parallel = Yes (Bind Degree = 4)

SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 145.198898
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)    Distribute Subsection #2
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
           |  #Columns = 5
```

```
        Subsection #1:
(    3)    Process Using 4 Subagents
(   10)    |  Access Table Queue  ID = q3  #Columns = 2
(    4)    |  Hash Join
           |  |  Estimated Build Size: 5737
           |  |  Estimated Probe Size: 8015
(    7)    |  |  Access Table Queue  ID = q5  #Columns = 3
(    5)    |  |  Hash Join
           |  |  |  Estimated Build Size: 5333
           |  |  |  Estimated Probe Size: 6421
(    6)    |  |  |  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
(    6)    |  |  |  |  Process Probe Table for Hash Join
(    3)    |  Insert Into Asynchronous Local Table Queue  ID = q2
(    3)    Access Local Table Queue  ID = q2  #Columns = 5
(    2)    Insert Into Asynchronous Table Queue  ID = q1
           |  Broadcast to Coordinator Node
           |  Rows Can Overflow to Temporary Table

        Subsection #2:
(   11)    Process Using 4 Subagents
(   12)    |  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
(   11)    |  Insert Into Asynchronous Local Table Queue  ID = q4
(   11)    Access Local Table Queue  ID = q4  #Columns = 2
(   10)    Insert Into Asynchronous Table Queue  ID = q3
           |  Hash to Specific Node
           |  Rows Can Overflow to Temporary Tables

        Subsection #3:
(    8)    Process Using 4 Subagents
(    9)    |  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
(    8)    |  Insert Into Asynchronous Local Table Queue  ID = q6
(    8)    Access Local Table Queue  ID = q6  #Columns = 3
(    7)    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)
                  |
                HSJOIN
                (   4)
               /      \
         HSJOIN        DTQ
         (   5)       (  10)
        /      \        |
    TBSCAN     DTQ     LTQ
    (   6)    (   7)  (  11)
      |         |       |
 Table:        LTQ    TBSCAN
 DOOLE        (   8)  (  12)
 DEPARTMENT     |       |
              TBSCAN  Table:
              (   9)  DOOLE
                |     PROJECT
            Table:
            DOOLE
            EMPLOYEE
```

This plan is similar to that in the third example, 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 the first example, 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"."EXAMPLE"  Version = ""

Prep Date = 2002/01/11
Prep Time = 13:52:48
```

```
Bind Timestamp = 2002-01-11-13.52.48.325413

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = No

SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION -------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 1804.625000
Estimated Cardinality = 112000.000000

(    7) Ship Distributed Subquery #2
        |  #Columns = 2
(    2) Hash Join
        |  Estimated Build Size: 48444
        |  Estimated Probe Size: 232571
(    6) |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
        |  |  #Columns = 3
        |  |  Relation Scan
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    6) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
        |  |  Estimated Build Size: 7111
        |  |  Estimated Probe Size: 64606
(    4) |  |  Ship Distributed Subquery #1
        |  |  |  #Columns = 3
(    1) Return Data to Application
        |  #Columns = 5

Distributed Substatement #1:
(    4) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

            SELECT A0."DEPTNO", A0."DEPTNAME", A0."LOCATION"
            FROM "DOOLE"."DEPARTMENT" A0
```

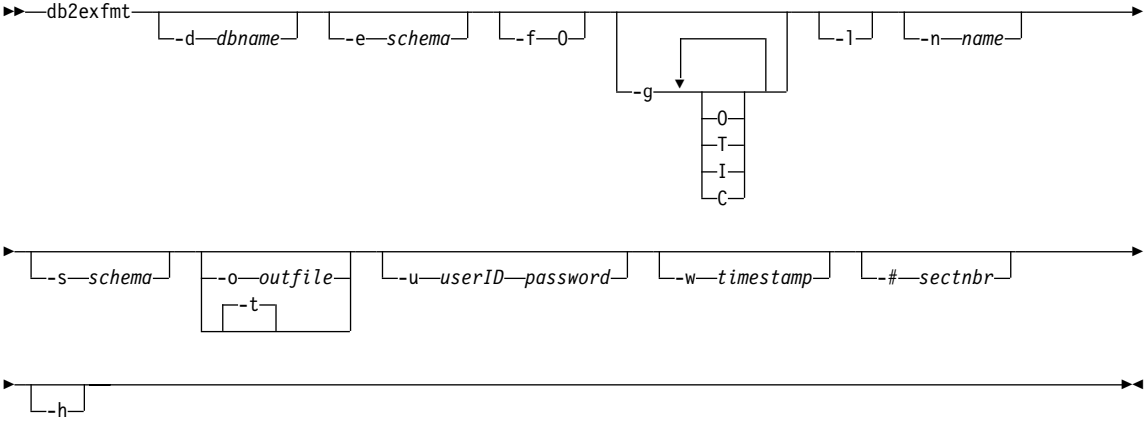
```
        Nicknames Referenced:
           DOOLE.DEPARTMENT  ID = 32768
              Base = DOOLE.DEPARTMENT
        #Output Columns = 3

Distributed Substatement #2:
(    7) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

             SELECT A0."DEPTNO", A0."PROJNAME"
             FROM "DOOLE"."PROJECT" A0

        Nicknames Referenced:
           DOOLE.PROJECT  ID = 32769
               Base = DOOLE.PROJECT
        #Output Columns = 2

End of section

Optimizer Plan:

                RETURN
                (   1)
                  |
                HSJOIN
                (   2)
               /      \
        HSJOIN           SHIP
        (   3)          (   7)
       /      \           |
    SHIP      TBSCAN   Nickname:
   (   4)     (   6)   DOOLE
     |          |      PROJECT
 Nickname:   Table:
 DOOLE       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.

Related concepts:

- "Explain tools" on page 228
- "Guidelines for using explain information" on page 230
- "Guidelines for capturing explain information" on page 239

Appendix E. DB2 Universal Database technical information

Overview of DB2 Universal Database technical information

DB2 Universal Database technical information can be obtained in the following formats:

- Books (PDF and hard-copy formats)
- A topic tree (HTML format)
- Help for DB2 tools (HTML format)
- Sample programs (HTML format)
- Command line help
- Tutorials

This section is an overview of the technical information that is provided and how you can access it.

Categories of DB2 technical information

The DB2 technical information is categorized by the following headings:

- Core DB2 information
- Administration information
- Application development information
- Business intelligence information
- DB2 Connect information
- Getting started information
- Tutorial information
- Optional component information
- Release notes

The following tables describe, for each book in the DB2 library, the information needed to order the hard copy, print or view the PDF, or locate the HTML directory for that book. A full description of each of the books in the DB2 library is available from the IBM Publications Center at www.ibm.com/shop/publications/order

The installation directory for the HTML documentation CD differs for each category of information:

htmlcdpath/doc/htmlcd/%L/*category*

where:

- *htmlcdpath* is the directory where the HTML CD is installed.
- *%L* is the language identifier. For example, `en_US`.
- *category* is the category identifier. For example, `core` for the core DB2 information.

In the PDF file name column in the following tables, the character in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d1e80` identifies the English version of the *Administration Guide: Planning* and the file name `db2d1g80` identifies the German version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

Language	**Identifier**
Arabic	w
Brazilian Portuguese	b
Bulgarian	u
Croatian	9
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Romanian	8
Russian	r
Simp. Chinese	c
Slovakian	7
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

No form number indicates that the book is only available online and does not have a printed version.

Core DB2 information

The information in this category cover DB2 topics that are fundamental to all DB2 users. You will find the information in this category useful whether you are a programmer, a database administrator, or you work with DB2 Connect, DB2 Warehouse Manager, or other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/core`.

Table 53. Core DB2 information

Name	Form Number	PDF File Name
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
IBM DB2 Universal Database Glossary	No form number	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Message Reference, Volume 1	GC09-4840	db2m1x80
IBM DB2 Universal Database Message Reference, Volume 2	GC09-4841	db2m2x80
IBM DB2 Universal Database What's New	SC09-4848	db2q0x80

Administration information

The information in this category covers those topics required to effectively design, implement, and maintain DB2 databases, data warehouses, and federated systems.

The installation directory for this category is `doc/htmlcd/%L/admin`.

Table 54. Administration information

Name	Form number	PDF file name
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80

Table 54. Administration information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guide to GUI Tools for Administration and Development	SC09-4851	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Installing and Administering a Satellite Environment	GC09-4823	db2dsx80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Application development information

The information in this category is of special interest to application developers or programmers working with DB2. You will find information about supported languages and compilers, as well as the documentation required to access DB2 using the various supported programming interfaces, such as embedded SQL, ODBC, JDBC, SQLj, and CLI. If you view this information online in HTML you can also access a set of DB2 sample programs in HTML.

The installation directory for this category is doc/htmlcd/%L/ad.

Table 55. Application development information

Name	Form number	PDF file name
IBM DB2 Universal Database Application Development Guide: Building and Running Applications	SC09-4825	db2axx80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

Business intelligence information

The information in this category describes how to use components that enhance the data warehousing and analytical capabilities of DB2 Universal Database.

The installation directory for this category is doc/htmlcd/%L/wareh.

Table 56. Business intelligence information

Name	Form number	PDF file name
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Installation Guide	GC27-1122	db2idx80

DB2 Connect information

The information in this category describes how to access host or iSeries data using DB2 Connect Enterprise Edition or DB2 Connect Personal Edition.

The installation directory for this category is `doc/htmlcd/%L/conn`.

Table 57. DB2 Connect information

Name	Form number	PDF file name
APPC, CPI-C, and SNA Sense Codes	No form number	db2apx80
IBM Connectivity Supplement	No form number	db2h1x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition	GC09-4833	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Getting started information

The information in this category is useful when you are installing and configuring servers, clients, and other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/start`.

Table 58. Getting started information

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Clients	GC09-4832	db2itx80
IBM DB2 Universal Database Quick Beginnings for DB2 Servers	GC09-4836	db2isx80
IBM DB2 Universal Database Quick Beginnings for DB2 Personal Edition	GC09-4838	db2i1x80
IBM DB2 Universal Database Installation and Configuration Supplement	GC09-4837	db2iyx80
IBM DB2 Universal Database Quick Beginnings for DB2 Data Links Manager	GC09-4829	db2z6x80

Tutorial information

Tutorial information introduces DB2 features and teaches how to perform various tasks.

The installation directory for this category is `doc/htmlcd/%L/tutr`.

Table 59. Tutorial information

Name	Form number	PDF file name
Business Intelligence Tutorial: Introduction to the Data Warehouse	No form number	db2tux80
Business Intelligence Tutorial: Extended Lessons in Data Warehousing	No form number	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	No form number	db2tdx80
Information Catalog Center Tutorial	No form number	db2aix80
Video Central for e-business Tutorial	No form number	db2twx80
Visual Explain Tutorial	No form number	db2tvx80

Optional component information

The information in this category describes how to work with optional DB2 components.

The installation directory for this category is `doc/htmlcd/%L/opt`.

Table 60. Optional component information

Name	Form number	PDF file name
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80

Table 60. Optional component information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6740	N/A

Release notes

The release notes provide additional information specific to your product's release and FixPak level. They also provides summaries of the documentation updates incorporated in each release and FixPak.

Table 61. Release notes

Name	Form number	PDF file name	HTML directory
DB2 Release Notes	See note.	See note.	doc/prodcd/*%L*/db2ir where *%L* is the language identifier.
DB2 Connect Release Notes	See note.	See note.	doc/prodcd/*%L*/db2cr where *%L* is the language identifier.
DB2 Installation Notes	Available on product CD-ROM only.	Available on product CD-ROM only.	

Note: The HTML version of the release notes is available from the Information Center and on the product CD-ROMs. To view the ASCII file:

- On UNIX-based platforms, see the `Release.Notes` file. This file is located in the `DB2DIR/Readme/%L` directory, where `%L` represents the locale name and `DB2DIR` represents:
  - `/usr/opt/db2_08_01` on AIX
  - `/opt/IBM/db2/V8.1` on all other UNIX operating systems
- On other platforms, see the `RELEASE.TXT` file. This file is located in the directory where the product is installed.

Related tasks:

• "Printing DB2 books from PDF files" on page 655

- "Ordering printed DB2 books" on page 656
- "Accessing online help" on page 656
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 660
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 661

Printing DB2 books from PDF files

You can print DB2 books from the PDF files on the *DB2 PDF Documentation* CD. Using Adobe Acrobat Reader, you can print either the entire book or a specific range of pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com

Procedure:

To print a DB2 book from a PDF file:

1. Insert the *DB2 PDF Documentation* CD. On UNIX operating systems, mount the DB2 PDF Documentation CD. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
   - On Windows operating systems:

     x:\doc*language* directory, where *x* represents the CD-ROM drive letter and *language* represents the two-character territory code that represents your language (for example, EN for English).
   - On UNIX operating systems:

     /cdrom/doc/*%L* directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

Related tasks:

- "Ordering printed DB2 books" on page 656
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 660
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 661

Related reference:

- "Overview of DB2 Universal Database technical information" on page 647

Ordering printed DB2 books

Procedure:

To order printed books:

- Contact your IBM authorized dealer or marketing representative. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/shop/planetwide
- Phone 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.
- Visit the IBM Publications Center at www.ibm.com/shop/publications/order

Related tasks:

- "Printing DB2 books from PDF files" on page 655
- "Finding topics by accessing the DB2 Information Center from a browser" on page 658
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 661

Related reference:

- "Overview of DB2 Universal Database technical information" on page 647

Accessing online help

The online help that comes with all DB2 components is available in three types:

- Window and notebook help
- Command line help
- SQL statement help

Window and notebook help explain the tasks that you can perform in a window or notebook and describe the controls. This help has two types:

- Help accessible from the **Help** button
- Infopops

The **Help** button gives you access to overview and prerequisite information. The infopops describe the controls in the window or notebook. Window and notebook help are available from DB2 centers and components that have user interfaces.

Command line help includes Command help and Message help. Command help explains the syntax of commands in the command line processor. Message help describes the cause of an error message and describes any action you should take in response to the error.

SQL statement help includes SQL help and SQLSTATE help. DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the syntax of SQL statements (SQL states and class codes).

Note: SQL help is not available for UNIX operating systems.

Procedure:

To access online help:

- For window and notebook help, click **Help** or click that control, then click **F1**. If the **Automatically display infopops** check box on the **General** page of the **Tool Settings** notebook is selected, you can also see the infopop for a particular control by holding the mouse cursor over the control.
- For command line help, open the command line processor and enter:
  - For Command help:

    ```
    ? command
    ```

    where *command* represents a keyword or the entire command.

    For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.
- For Message help:

  ```
  ? XXXnnnnn
  ```

  where *XXXnnnnn* represents a valid message identifier.

  For example, `? SQL30081` displays help about the SQL30081 message.
- For SQL statement help, open the command line processor and enter:
  - For SQL help:

    ```
    ? 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.
  - For SQLSTATE help:

```
help statement
```

where *statement* represents an SQL statement.

For example, `help SELECT` displays help about the SELECT statement.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 658
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 661

Finding topics by accessing the DB2 Information Center from a browser

The DB2 Information Center accessed from a browser enables you to access the information you need to take full advantage of DB2 Universal Database and DB2 Connect. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, metadata, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser is composed of the following major elements:

Navigation tree
The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topics, the glossary, and the master index in the DB2 Information Center.

Navigation toolbar
The navigation toolbar is located in the top right frame of the browser window. The navigation toolbar contains buttons that enable you to search the DB2 Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame
The content frame is located in the bottom right frame of the browser window. The content frame displays topics from the DB2 Information Center when you click on a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index.

Prerequisites:

To access the DB2 Information Center from a browser, you must use one of the following browsers:

- Microsoft Explorer, version 5 or later
- Netscape Navigator, version 6.1 or later

Restrictions:

The DB2 Information Center contains only those sets of topics that you chose to install from the *DB2 HTML Documentation CD*. If your Web browser returns a `File not found` error when you try to follow a link to a topic, you must install one or more additional sets of topics *DB2 HTML Documentation CD*.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon of the category of topics related to your area of interest. A list of subcategories displays underneath the icon.
2. Continue to click the book icons until you find the category containing the topics in which you are interested. Categories that link to topics display the category title as an underscored link when you move the cursor over the category title. The navigation tree identifies topics with a page icon.
3. Click the topic link. The topic displays in the content frame.

To find a topic or term in the master index:

1. In the navigation tree, click the “Index” category. The category expands to display a list of links arranged in alphabetical order in the navigation tree.
2. In the navigation tree, click the link corresponding to the first character of the term relating to the topic in which you are interested. A list of terms with that initial character displays in the content frame. Terms that have multiple index entries are identified by a book icon.
3. Click the book icon corresponding to the term in which you are interested. A list of subterms and topics displays below the term you clicked. Topics are identified by page icons with an underscored title.
4. Click on the title of the topic that meets your needs. The topic displays in the content frame.

Related concepts:

- "Accessibility" on page 667
- "DB2 Information Center for topics" on page 669

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 660
- "Updating the HTML documentation installed on your machine" on page 662
- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 664
- "Searching the DB2 documentation" on page 665

Related reference:

- "Overview of DB2 Universal Database technical information" on page 647

Finding product information by accessing the DB2 Information Center from the administration tools

The DB2 Information Center provides quick access to DB2 product information and is available on all operating systems for which the DB2 administration tools are available.

The DB2 Information Center accessed from the tools provides six types of information.

Tasks Key tasks you can perform using DB2.

Concepts
Key concepts for DB2.

Reference
DB2 reference information, such as keywords, commands, and APIs.

Troubleshooting
Error messages and information to help you with common DB2 problems.

Samples
Links to HTML listings of the sample programs provided with DB2.

Tutorials
Instructional aid designed to help you learn a DB2 feature.

Prerequisites:

Some links in the DB2 Information Center point to Web sites on the Internet. To display the content for these links, you will first have to connect to the Internet.

Procedure:

To find product information by accessing the DB2 Information Center from the tools:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Click the tab of the information type related to the information you are attempting to find.
3. Navigate through the tree and click on the topic in which you are interested. The Information Center will then launch a Web browser to display the information.
4. To find information without browsing the lists, click the **Search** icon to the right of the list.

   Once the Information Center has launched a browser to display the information, you can perform a full-text search by clicking the **Search** icon in the navigation toolbar.

Related concepts:

- "Accessibility" on page 667
- "DB2 Information Center for topics" on page 669

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 658
- "Searching the DB2 documentation" on page 665

Viewing technical documentation online directly from the DB2 HTML Documentation CD

All of the HTML topics that you can install from the *DB2 HTML Documentation CD* can also be read directly from the CD. Therefore, you can view the documentation without having to install it.

Restrictions:

Because the following items are installed from the DB2 product CD and not the *DB2 HTML Documentation CD*, you must install the DB2 product to view these items:

- Tools help
- DB2 Quick Tour
- Release notes

Procedure:

1. Insert the *DB2 HTML Documentation* CD. On UNIX operating systems, mount the *DB2 HTML Documentation CD*. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start your HTML browser and open the appropriate file:
   - For Windows operating systems:

     ```
     e:\Program Files\sqllib\doc\htmlcd\%L\index.htm
     ```

     where *e* represents the CD-ROM drive, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.
   - For UNIX operating systems:

     ```
     /cdrom/Program Files/sqllib/doc/htmlcd/%L/index.htm
     ```

     where */cdrom/* represents where the CD is mounted, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 658
- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 664

Related reference:

- "Overview of DB2 Universal Database technical information" on page 647

Updating the HTML documentation installed on your machine

It is now possible to update the HTML installed from the *DB2 HTML Documentation CD* when updates are made available from IBM. This can be done in one of two ways:

- Using the Information Center (if you have the DB2 administration GUI tools installed).
- By downloading and applying a DB2 HTML documentation FixPak .

Note: This will NOT update the DB2 code; it will only update the HTML documentation installed from the *DB2 HTML Documentation CD*.

Procedure:

To use the Information Center to update your local documentation:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Ensure your machine has access to the external Internet; the updater will download the latest documentation FixPak from the IBM server if required.
3. Select **Information Center** —> **Update Local Documentation** from the menu to start the update.
4. Supply your proxy information (if required) to connect to the external Internet.

The Information Center will download and apply the latest documentation FixPak, if one is available.

To manually download and apply the documentation FixPak :

1. Ensure your machine is connected to the Internet.
2. Open the DB2 support page in your Web browser at: www.ibm.com/software/data/db2/udb/winos2unix/support.
3. Follow the link for version 8 and look for the ″Documentation FixPaks″ link.
4. Determine if the version of your local documentation is out of date by comparing the documentation FixPak level to the documentation level you have installed. This current documentation on your machine is at the following level: **DB2 v8.1 GA**.
5. If there is a more recent version of the documentation available then download the FixPak applicable to your operating system. There is one FixPak for all Windows platforms, and one FixPak for all UNIX platforms.
6. Apply the FixPak:
   - For Windows operating systems: The documentation FixPak is a self extracting zip file. Place the downloaded documentation FixPak in an empty directory, and run it. It will create a **setup** command which you can run to install the documentation FixPak.
   - For UNIX operating systems: The documentation FixPak is a compressed tar.Z file. Uncompress and untar the file. It will create a directory named `delta_install` with a script called **installdocfix**. Run this script to install the documentation FixPak.

Related tasks:

- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 664

Related reference:

- "Overview of DB2 Universal Database technical information" on page 647

Copying files from the DB2 HTML Documentation CD to a Web Server

The entire DB2 information library is delivered to you on the *DB2 HTML Documentation CD*, so you can install the library on a Web server for easier access. Simply copy to your Web server the documentation for the languages that you want.

Procedure:

To copy files from the *DB2 HTML Documentation CD* to a Web server, use the appropriate path:

- For Windows operating systems:

  E:\Program Files\sqllib\doc\htmlcd*%L**.*

  where *E* represents the CD-ROM drive and *%L* represents the language identifier.
- For UNIX operating systems:

  /*cdrom*:Program Files/sqllib/doc/htmlcd/*%L*/*.*

  where *cdrom* represents the CD-ROM drive and *%L* represents the language identifier.

Related tasks:

- "Searching the DB2 documentation" on page 665

Related reference:

- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "Overview of DB2 Universal Database technical information" on page 647

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by web browsers. This task describes possible workarounds.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following workaround, especially if you see the following line in the browser Java console:

```
Cannot find class  java/security/InvalidParameterException
```

- On Windows operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied *x*:Program Files\sqllib\doc\htmlcd*locale*\InvalidParameterException.class file to the java\classes\java\security\ directory relative to your Netscape browser installation, where *x* represents the CD-ROM drive letter and *locale* represents the name of the desired locale.

  Note: You may have to create the java\security\ subdirectory structure.

- On UNIX operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied */cdrom*/Program Files/sqllib/doc/htmlcd/*locale*/InvalidParameterException.class file to the java/classes/java/security/ directory relative to your Netscape browser installation, where *cdrom* represents the mount point of the CD-ROM and *locale* represents the name of the desired locale.

  Note: You may have to create the java/security/ subdirectory structure.

If your Netscape browser still fails to display the search input window, try the following:

- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the machine. Then open a new instance of the Netscape browser and try to start the search again.
- Purge the browser's cache.
- Try a different version of Netscape, or a different browser.

Related tasks:

- "Searching the DB2 documentation" on page 665

Searching the DB2 documentation

To search DB2's documentation, you need Netscape 6.1 or higher, or Microsoft's Internet Explorer 5 or higher. Ensure that your browser's Java support is enabled.

A pop-up search window opens when you click the search icon in the navigation toolbar of the Information Center accessed from a browser. If you are using the search for the first time it may take a minute or so to load into the search window.

Restrictions:

The following restrictions apply when you use the documentation search:

- Boolean searches are not supported. The boolean search qualifiers *and* and *or* will be ignored in a search. For example, the following searches would produce the same results:
  - servlets *and* beans
  - servlets *or* beans
- Wildcard searches are not supported. A search on *java** will only look for the literal string *java** and would not, for example, find *javadoc*.

In general, you will get better search results if you search for phrases instead of single words.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

Note: When you perform a search, the first result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in results lists.

Related tasks:

- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 664

Online DB2 troubleshooting information

With the release of DB2® UDB Version 8, there will no longer be a *Troubleshooting Guide*. The troubleshooting information once contained in this guide has been integrated into the DB2 publications. By doing this, we are able to deliver the most up-to-date information possible. To find information on the troubleshooting utilities and functions of DB2, access the DB2 Information Center from any of the tools.

Refer to the DB2 Online Support site if you are experiencing problems and want help finding possible causes and solutions. The support site contains a

large, constantly updated database of DB2 publications, TechNotes, APAR (product problem) records, FixPaks, and other resources. You can use the support site to search through this knowledge base and find possible solutions to your problems.

Access the Online Support site at www.ibm.com/software/data/db2/udb/winos2unix/support, or by clicking the **Online Support** button in the DB2 Information Center. Frequently changing information, such as the listing of internal DB2 error codes, is now also available from this site.

Related concepts:

- "DB2 Information Center for topics" on page 669

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 660

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in DB2® Universal Database Version 8:

- DB2 allows you to operate all features using the keyboard instead of the mouse. See "Keyboard Input and Navigation".
- DB2 enables you customize the size and color of your fonts. See "Accessible Display" on page 668.
- DB2 allows you to receive either visual or audio alert cues. See "Alternative Alert Cues" on page 668.
- DB2 supports accessibility applications that use the Java™ Accessibility API. See "Compatibility with Assistive Technologies" on page 668.
- DB2 comes with documentation that is provided in an accessible format. See "Accessible Documentation" on page 668.

Keyboard Input and Navigation

Keyboard Input

You can operate the DB2 Tools using only the keyboard. You can use keys or key combinations to perform most operations that can also be done using a mouse.

Keyboard Focus

In UNIX-based systems, the position of the keyboard focus is highlighted, indicating which area of the window is active and where your keystrokes will have an effect.

Accessible Display

The DB2 Tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font Settings

The DB2 Tools allow you to select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

Non-dependence on Color

You do not need to distinguish between colors in order to use any of the functions in this product.

Alternative Alert Cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with Assistive Technologies

The DB2 Tools interface supports the Java Accessibility API enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible Documentation

Documentation for the DB2 family of products is available in HTML format. This allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

DB2 tutorials

The DB2® tutorials help you learn about various aspects of DB2 Universal Database. The tutorials provide lessons with step-by-step instructions in the areas of developing applications, tuning SQL query performance, working with data warehouses, managing metadata, and developing Web services using DB2.

Before you begin:

Before you can access these tutorials using the links below, you must install the tutorials from the *DB2 HTML Documentation* CD-ROM.

If you do not want to install the tutorials, you can view the HTML versions of the tutorials directly from the *DB2 HTML Documentation CD*. PDF versions of these tutorials are also available on the *DB2 PDF Documentation CD*.

Some tutorial lessons use sample data or code. See each individual tutorial for a description of any prerequisites for its specific tasks.

DB2 Universal Database tutorials:

If you installed the tutorials from the *DB2 HTML Documentation* CD-ROM, you can click on a tutorial title in the following list to view that tutorial.

Business Intelligence Tutorial: Introduction to the Data Warehouse Center
Perform introductory data warehousing tasks using the Data Warehouse Center.

Business Intelligence Tutorial: Extended Lessons in Data Warehousing
Perform advanced data warehousing tasks using the Data Warehouse Center.

Development Center Tutorial for Video Online using Microsoft® Visual Basic
Build various components of an application using the Development Center Add-in for Microsoft Visual Basic.

Information Catalog Center Tutorial
Create and manage an information catalog to locate and use metadata using the Information Catalog Center.

Video Central for e-business Tutorial
Develop and deploy an advanced DB2 Web Services application using WebSphere® products.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 Information Center for topics

The DB2® Information Center gives you access to all of the information you need to take full advantage of DB2 Universal Database™ and DB2 Connect™ in your business. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, the Information Catalog Center, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser has the following features:

Regularly updated documentation
Keep your topics up-to-date by downloading updated HTML.

Search

Search all of the topics installed on your workstation by clicking **Search** in the navigation toolbar.

Integrated navigation tree

Locate any topic in the DB2 library from a single navigation tree. The navigation tree is organized by information type as follows:

- Tasks provide step-by-step instructions on how to complete a goal.
- Concepts provide an overview of a subject.
- Reference topics provide detailed information about a subject, including statement and command syntax, message help, requirements.

Master index

Access the information in topics and tools help from one master index. The index is organized in alphabetical order by index term.

Master glossary

The master glossary defines terms used in the DB2 Information Center. The glossary is organized in alphabetical order by glossary term.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 658
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 660
- "Updating the HTML documentation installed on your machine" on page 662

Appendix F. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

access plans
  effect on locks 85
  for column correlation with multiple predicates 174
  methods 176
  using indexes 31, 177
accessibility
  features 667
ADVISE_INDEX table 597
ADVISE_WORKLOAD table 600
agent pool size configuration parameter 448
agent process
  applheapsz configuration parameter 410
  aslheapsz configuration parameter 417
  maximum number of agents 444
  maximum number of concurrent agents 445
  priority of agents configuration parameter 443
agent_stack_sz configuration parameter 412
agentpri configuration parameter 443
agents
  client connections 312
  configuration parameters affecting number of 311
  described 308
  in a partitioned database 315
  managing 310
  memory use 258
  worker agent types 308
ALTER TABLESPACE statement
  example 111
app_ctl_heap_sz 403
APPC transaction program name configuration parameter 498
APPEND mode, insert process for 34
appgroup_mem_sz configuration parameter 402
APPLHEAPSZ configuration parameter
  usage 410
application control heap size configuration parameter 403
application global memory, configuring parameters for 258
application process
  effect on locks 84
application program 51
application support layer heap size configuration parameter 417
applications
  control heap, setting 403
  maximum number of coordinating agents at node 446
  shared memory use 258
aslheapsz configuration parameter 417
audit_buf_sz configuration parameter 425
authentication
  trust all clients configuration parameter 527
  Use SNA Authentication configuration parameter 525
authentication configuration parameter 523
authentication DAS configuration parameter 537
authority
  configuration parameters 520
auto restart enable configuration parameter 470
automatic configuration parameters 371
automatic summary tables
  description 210
avg_appls configuration parameter 440

B

backup pending indicator configuration parameter 487
backup_pending configuration parameter 487
backups
  track modified pages 474
benchmarking
  db2batch tool 358
  overview 355
  preparation for 356
benchmarking *(continued)*
  sample report 366
  SQL statements for 356
  steps summarized 364
  testing methods 355
  testing process 364
binding
  changing configuration parameters 372
  isolation level 57
blk_log_dsk_ful configuration parameter 469
block on log disk full (blk_log_dsk_ful) configuration parameter 469
block-based buffer pools
  for prefetching efficiency 277
buffer pools
  block-based, prefetching performance 277
  data-page management in 269
  effect on query optimization 163
  how used 264
  large, advantage of 271
  memory allocation at startup 271
  multiple
    advantages of 271
    managing 271
    pages sizes for 271
  page cleaners, tuning 267
  secondary 266
bypass federated authentication configuration parameter 525

C

capacity
  methods of expanding 337
capacity management configuration parameters 391
catalog cache size configuration parameter 393
catalog node 51
catalog statistics
  catalog table descriptions 124
  collecting
    distribution statistics on specific columns 121
    index statistics 123

catalog statistics *(continued)*
collecting *(continued)*
requirements and method described 120
updating 119
detailed index data collected 142
distribution statistics
extended example of use 138
frequency 133
quantile 133
when to collect 133
for sub-elements in columns 144
for user-defined functions 146
how used 117
index cluster ratio 183
information collected 131
manual adjustments for modeling 147
manual update guidelines 152
manual update rules
column statistics 153
distribution 153
index statistics 155
table and nickname 154
modeling production databases with 149
when collected 117
catalog tables
description 124
catalog_noauth configuration parameter 526
catalogcache_sz configuration parameter 393
change the database log path configuration parameter 459
character conversion
effect on application performance 105
chngpgs_thresh configuration parameter 431
client I/O block size configuration parameter 420
client support
client I/O block size configuration parameter 420
TCP/IP service name configuration parameter 497
tpname configuration parameter 498
clustering indexes 23
code pages
database configuration parameter 482
codepage configuration parameter 482
codeset configuration parameter 482
collate_info configuration parameter 483
columns
collecting distribution statistics on specific 121
subelements, collecting statistics for 144
updating statistics manually, rules 153
comm_bandwidth configuration parameter
description 513
effect on query optimization 163
commit
number of commits to group (mincommit) 464
communications
connection elapse time 501
compilers
rewrites
adding implied predicates 173
correlated subqueries 171
merge view 168
compound SQL
how used 103
concurrency control
for federated databases 51
general issues for 51
maximum number of active applications 438
maximum number of concurrently active databases 514
configuration
parameter details 390
configuration file release level configuration parameter 481
configuration files
description 369
location 369
configuration parameters
affecting number of agents 311
affecting query optimization 163
agent communication memory 416
agent private memory 405
agent_stack_sz 412
agentpri 443
app_ctl_heap_sz 403
appgroup_mem_sz 402
configuration parameters *(continued)*
applheapsz 410
application communication memory 416
application shared memory 401
applications and agents 438
aslheapsz 417
audit_buf_sz 425
authentication 523
authentication (DAS) 537
automatic 371
autorestart 470
avg_appls 440
backup_pending 487
blk_log_dsk_ful 469
capacity management 391
catalog_noauth 526
catalogcache_sz 393
chngpgs_thresh 431
codepage 482
codeset 482
collate_info 483
comm_bandwidth 513
communication protocol setup 496
communications 496
compiler settings 489
conn_elapse 501
contact_host 536
cpuspeed 513
das_codepage 537
das_territory 538
dasadm_group 531
database attributes 481
database management 480
database manager instance memory 421
database shared memory 391
database status 486
database system monitor 511
database_consistent 487
database_level 481
database_memory 391
datalinks 486
DB2 Data Links Manager 483
DB2 discovery 498
db2system 530
dbheap 392
description 369
dft_account_str 518
dft_degree 491
dft_extent_sz 436
dft_loadrec_ses 472
dft_monswitches 511
dft_prefetch_sz 435

configuration parameters *(continued)*
dft_queryopt 491
dft_refresh_age 492
dft_sqlmathwarn 489
dftdbpath 526
diaglevel 507
diagnostic information 507
diagpath 508
dir_cache 423
discover 499
discover (DAS) 530
discover_comm 499
discover_db 498
discover_inst 500
distributed unit of work 476
dl_expint 484
dl_num_copies 485
dl_time_drop 485
dl_token 485
dl_upper 486
dl_wt_iexpint 484
dlchktime 427
dyn_query_mgmt 480
estore_seg_sz 44, 437
exec_exp_task 535
fcm_num_buffers 502
fed_noauth 525
federated 520
fenced_pool 452
groupheap_ratio 403
health_mon 510
I/O and storage 431
indexrec 470
instance administration 520
instance management 507
instance_memory 421
intra_parallel 506
java_heap_sz 426
jdk_path 519
jdk_path (DAS) 535
keepfenced 36, 450
locklist 397
locks 427
locktimeout 430
log activity 464
log files 454
log_retain_status 488
logbufsz 395
logfilsiz 454
logging 454
loghead 464
logpath 463
logprimary 456
logretain 467
logsecond 458
configuration parameters *(continued)*
max_connections 44, 448
max_connretries 503
max_coordagents 446
max_querydegree 505
max_time_diff 504
maxagents 44, 444
maxappls 438
maxcagents 445
maxfilop 441
maxlocks 428
maxtotfilop 442
min_dec_div_3 419
min_priv_mem 414
mincommit 464
mirrorlogpath 461
mon_heap_sz 422
multipage_alloc 488
newlogpath 459
nname 496
nodetype 518
notifylevel 509
num_db_backups 473
num_estore_segs 44, 437
num_freqvalues 493
num_initagents 450
num_initfenced 453
num_iocleaners 432
num_ioservers 434
num_poolagents 448
num_quantiles 494
numdb 44, 514
numsegs 436
overflowlogpath 462
parallel operations 501
partitioned database 501
pckcachesz 399
priv_mem_thresh 415
query enabler 480
query_heap_sz 411
rec_his_retentn 473
recovery 454, 469
release 481
restore_pending 488
resync_interval 477
rollfwd_pending 487
rqrioblk 420
sched_enable 532
sched_userid 536
seqdetect 434
sheapthres 406
sheapthres_shr 408
smtp_server 534
softmax 465
sortheap 405
configuration parameters *(continued)*
spm_log_file_sz 479
spm_log_path 478
spm_max_resync 479
spm_name 478
start_stop_time 504
stat_heap_sz 410
stmtheap 409
stored procedures 450
svcename 497
sysadm_group 521
sysctrl_group 522
sysmaint_group 523
system management 512
territory 482
Tivoli Storage Manager 469
tm_database 476
toolscat_db 533
toolscat_inst 532
toolscat_schema 533
tp_mon_name 516
tpname 498
trackmod 474
trust_allclnts 527
trust_clntauth 528
tsm_mgmtclass 474
tsm_nodename 475
tsm_owner 476
tsm_password 475
use_sna_auth 525
user defined functions 450
user_exit_status 488
userexit 468
util_heap_sz 396
configurations
changing database parameters 372
parameter summary, database 376
parameter summary, database manager 376
tuning parameters 371
conn_elapse configuration parameter 501
connection concentrator
client-connection improvements 312
usage examples 312
use of agents in partitioned database 315
connection elapse time configuration parameter 501
connections
elapse time 501

constraints
Explain tables 577
contact_host configuration parameter 536
Control Center
Event Analyzer 316
Performance Monitor 316
Snapshot Monitor 316
Coordinated Universal Time 504
coordinator agent
connection-concentrator use 312
how used 36
cpuspeed configuration parameter
description 513
effect on query optimization 163
CURRENT EXPLAIN MODE special register
capturing explain data 239
CURRENT EXPLAIN SNAPSHOT special register
capturing explain information 239

D

DAS configuration parameters 529
authentication 537
contact_host 536
das_codepage 537
das_territory 538
dasadm_group 531
db2system 530
exec_exp_task 535
jdk_path 535
sched_enable 532
sched_userid 536
smtp_server 534
toolscat_db 533
toolscat_inst 532
toolscat_schema 533
das_codepage configuration parameter 537
das_territory configuration parameter 538
dasadm_group configuration parameter 531
Data Links access token expiry interval configuration parameter 484
Data Links number of copies configuration parameter 485
Data Links time after drop configuration parameter 485
Data Links token algorithm configuration parameter 485
Data Links token in upper case configuration parameter 486
data page in standard tables 23
data redistribution
determining need for 349
error recovery 353
guidelines for 347
instructions for 350
log space requirements for 352
process description 347
data sources
I/O speed and performance 220
data-stream information
displayed by db2expln 622
database 51
database directories
structure described 16
database global memory configuration parameters 258
database heap configuration parameter 392
database management, configuration parameters 480
database manager 51
configuration parameter summary 376
machine node type configuration parameter 518
shared memory use 254
start timeout 504
stop timeout 504
database monitor
using 316
database partitions
adding 338
database shared memory size configuration parameter 391
database system monitor configuration parameters 511
database territory code configuration parameter 482
database_consistent configuration parameter 487
database_level configuration parameter 481
database_memory configuration parameter 391
database-managed space (DMS)
description 20
table-space address map 22
databases
autorestart configuration parameter 470
backup_pending configuration parameter 487
databases *(continued)*
codepage configuration parameter 482
codeset configuration parameter 482
collating information 483
configuration parameter summary 376
maximum number of concurrently active databases 514
release level configuration parameter 481
territory code configuration parameter 482
territory configuration parameter 482
DATALINK data type configuration parameter 486
DB2 architecture overview 11
DB2 Data Links Manager 483
DB2 documentation search
using Netscape 4.x 664
DB2 Information Center 669
DB2 tutorials 668
DB2_ANTIJOIN 556
DB2_APM_PERFORMANCE 562
DB2_AVOID_PREFETCH 562
DB2_AWE 562
DB2_BINSORT 562
DB2_CORRELATED_PREDICATES 556
DB2_ENABLE_BUFPD 562
DB2_EXTENDED_OPTIMIZATION 562
DB2_HASH_JOIN 556
DB2_INLIST_TO_NLJN 556
DB2_LIKE_VARCHAR 556
DB2_MINIMIZE_LIST_PREFETCH 556
DB2_MMAP_READ 562
DB2_MMAP_WRITE 562
DB2_NEW_CORR_SQ_FF 556
DB2_OVERRIDE_BPF 562
DB2_PINNED_BP 562
DB2_PRED_FACTORIZE 556
DB2_REDUCED_OPTIMIZATION 556
DB2_SELECTIVITY 556
DB2_SORT_AFTER_TQ 562
DB2_STPROC_LOOKUP_FIRST 562
DB2_USE_PAGE_CONTAINER_TAG 546
DB2ACCOUNT registry variable 542
DB2ADMINSERVER 571
DB2ATLDPORTS 554
DB2ATLDPWFILE 554

db2batch benchmarking tool
creating tests 358
examples 360
DB2BIDI registry variable 542
DB2BPVARS 562
DB2BQTIME 553
DB2BQTRY 553
DB2CHECKCLIENTINTERVAL registry variable 548
DB2CHGPWDESE 554
DB2CHKPTR 562
DB2CLIINIPATH 571
DB2CODEPAGE registry variable 542
DB2COMM registry variable 548
DB2CONNECTINAPPPROCESS environment variable 546
DB2DARILOOKUPALL 562
DB2DBDFT registry variable 542
DB2DBMSADDR registry variable 542
DB2DEFPREP 571
DB2DISABLEFLUSHLOG registry variable 542
DB2DISCOVERYTIME registry variable 542
DB2DJCOMM 571
DB2DMNBCKCTLR 571
DB2DOMAINLIST 546
db2empfa command 19
DB2ENABLELDAP 571
DB2ENVLIST 546
db2exfmt tool 645
db2expln tool
block and RID preparation information 624
information displayed
aggregation 625
data stream 622
insert, update, delete 623
join 620
miscellaneous 630
parallel processing 626
table access 611
temporary table 617
output 610
output samples
description 633
for federated database plan 642
multipartition plan with full parallelism 640
multipartition plan with inter-partition parallelism 637
db2expln tool *(continued)*
output samples *(continued)*
no parallelism 633
single-partition plan with intra-partition parallelism 635
syntax and parameters 602
usage notes 608
DB2FALLBACK 571
DB2FORCEFCMBP 554
DB2FORCENLSCACHE registry variable 548
DB2GRPLOOKUP 571
DB2INCLUDE registry variable 542
DB2INSTANCE environment variable 546
DB2INSTDEF registry variable 542
DB2INSTOWNER registry variable 542
DB2INSTPROF environment variable 546
DB2IQTIME 553
DB2LDAPBASEDN 571
DB2LDAPCACHE 571
DB2LDAPCLIENTPROVIDER 571
DB2LDAPHOST 571
DB2LDAPSEARCHSCOPE 571
DB2LIBPATH 546
DB2LICSTATSIZE registry variable registry variable 542
DB2LOADREC registry variable 571
DB2LOCALE registry variable 542
DB2LOCKTORB registry variable 571
DB2MAXFSCRSEARCH performance variable 562
DB2MEMDISCLAIM performance variable 562
DB2MEMMAXFREE performance variable 562
DB2NBADAPTERS registry variable 548
DB2NBBRECVNCBS registry variable 548
DB2NBCHECKUPTIME registry variable 548
DB2NBDISCOVERRCVBUFS registry variable 542
DB2NBINTRLISTENS registry variable 548
DB2NBRECVBUFFSIZE registry variable 548
DB2NBRESOURCES registry variable 548
DB2NBSENDNCBS registry variable 548
DB2NBSESSIONS registry variable 548
DB2NBXTRANCBS registry variable 548
DB2NETREQ registry variable 548
DB2NEWLOGPATH2 registry variable 571
DB2NODE 546
exported when adding server 339, 341, 342
DB2NOEXITLIST 571
DB2NTMEMSIZE 562
DB2NTPRICLASS 562
DB2NTWORKSET 562
DB2NUMFAILOVERNODES 554
DB2OPTIONS registry variable 542
DB2PARALLELIO 546
DB2PATH environment variable 546
DB2PORTRANGE 554
DB2PRIORITIES 562
DB2REMOTEPREG 571
DB2RETRY registry variable 548
DB2RETRYTIME registry variable 548
DB2ROUTINEDEBUG 571
DB2RQTIME 553
DB2SERVICETPINSTANCE registry variable 548
DB2SLOGON registry variable 542
DB2SORCVBUF 571
DB2SORT 571
DB2SOSNDBUF registry variable 548
DB2SYSPLEXSERVER registry variable 548
DB2SYSTEM 571
db2system configuration parameter 530
DB2TCPCONNMGRS registry variable 548
DB2TERRITORY registry variable 542
DB2TIMEOUT registry variable 542
DB2TRACEFLUSH registry variable 542
DB2TRACENAME registry variable 542
DB2TRACEON registry variable 542
DB2TRCSYSERR registry variable 542
DB2VENDORINI 571

DB2VIDEVICE registry
variable 548
DB2VIENABLE registry
variable 548
DB2VIVIPL registry variable 548
DB2XBSALIBRARY 571
DB2YIELD registry variable 542
DBHEAP configuration
parameter 392
deadlock detector
described 14
deadlocks
checking for 427
described 14
dlchktime configuration
parameter 427
decimal arithmetic
decimal division scale to 3
configuration parameter 419
decorrelation of a query
compiler rewrites for 171
default database path configuration
parameter 526
default number of SMS containers
configuration parameter 436
defragmentation
index 305
dft_account_str configuration
parameter 518
dft_degree configuration
parameter 491
effect on query optimization 163
dft_extent_sz configuration
parameter 436
dft_loadrec_ses configuration
parameter 472
dft_mon_bufpool configuration
parameter 511
dft_mon_lock configuration
parameter 511
dft_mon_sort configuration
parameter 511
dft_mon_stmt configuration
parameter 511
dft_mon_table configuration
parameter 511
dft_mon_timestamp configuration
parameter 511
dft_mon_uow configuration
parameter 511
dft_monswitches configuration
parameter 511
dft_prefetch_sz configuration
parameter 435
dft_queryopt configuration
parameter 491
dft_refresh_age configuration
parameter 492
dft_sqlmathwarn configuration
parameter 489
dftdbpath configuration
parameter 526
diaglevel configuration
parameter 507
diagpath configuration
parameter 508
dir_cache configuration
parameter 423
directory cache support
configuration parameter 423
disability 667
discover (DAS) configuration
parameter 530
discover configuration
parameter 499
discover server instance
configuration parameter 500
discover_comm configuration
parameter 499
discover_db configuration
parameter 498
discover_inst configuration
parameter 500
discovery mode configuration
parameter 499
disks
storage performance factors 15
distribution statistics
described 133
extended example of use 138
manual update rules 153
optimizer use of 136
dl_expint configuration
parameter 484
dl_num_copies configuration
parameter 485
dl_time_drop configuration
parameter 485
dl_token configuration
parameter 485
dl_upper configuration
parameter 486
dl_wt_iexpint configuration
parameter 484
dlchktime configuration
parameter 427
DLFM_BACKUP_DIR_NAME
variable 569
DLFM_BACKUP_LOCAL_MP
variable 569
DLFM_BACKUP_TARGET
_LIBRARY variable 569
DLFM_BACKUP_TARGET
variable 569
DLFM_ENABLE_STPROC
variable 569
DLFM_FS_ENVIRONMENT
variable 569
DLFM_GC_MODE variable 569
DLFM_INSTALL_PATH
variable 569
DLFM_LOG_LEVEL variable 569
DLFM_PORT variable 569
DLFM_TSM_MGMTCLASS
variable 569
DMS device
buffering behavior for 307
caching behavior 307
dyn_query_mgmt configuration
parameter 480
dynamic SQL
setting optimization class 93
dynexpln tool
output described 610
syntax and parameters 610

E

enable Data Links support
configuration parameter 486
enable intra-partition parallelism
configuration parameter 506
engine dispatchable unit (EDU)
agents 308
description 44
environment variables
overview 541
error handling
configuration parameters 507
error messages
when adding nodes to
partitioned databases 344
estore_seg_sz configuration
parameter 437
for memory management 44
event snapshots
described 316
exec_exp_task configuration
parameter 535
execute expired tasks configuration
parameter 535
Explain facility
analyzing information from 241
capturing information with 239

Explain facility *(continued)*
described 227
information displayed
for data objects 234
for data operators 234
for instances 235
snapshots, creating 239
using collected information 230
explain instance
defined 232
explain tables
formatting tool for data in 645
organization described 232
overview 577
explain tools
db2exfmt described 228
db2expln described 228
dynexpln described 228
summarized 228
using 601
Visual Explain described 228
EXPLAIN_ARGUMENT table
description 578
EXPLAIN_INSTANCE table
description 582
EXPLAIN_OBJECT table
description 585
EXPLAIN_OPERATOR table
description 588
EXPLAIN_PREDICATE table
description 590
EXPLAIN_STATEMENT table
description 592
EXPLAIN_STREAM table
description 595
extended storage
described 44
extent
extent map pages (EMP)for DMS table spaces 20
for SMS table spaces 19

F

fast communications manager (FCM)
description 44
FCM buffer pool
illustration of 256
memory requirements 256
FCM buffers configuration parameter 502
fcm_num_buffers configuration parameter 502
fed_noauth configuration parameter 525
federated configuration parameter 520
federated databases
analyzing where queries evaluated 218
compiler phases 213
concurrency control for 51
db2expln output for query in 642
effect of server options on 115
global analysis of queries on 223
global optimization in 220
pushdown analysis 213
query information 629
system support configuration parameter 520
fenced_pool configuration parameter 452
first active log file configuration parameter 464
FOR FETCH ONLY clause
in query tuning 95
FOR READ ONLY clause
in query tuning 95
free space control record (FSCR)
in MDC tables 28
in standard tables 23

G

governor tool
configuration file example 331
configuration file rule descriptions 324
configuring 323
daemon described 321
described 319
log files created by 333
queries against log files 334
rule elements 326
starting and stopping 320
groupheap_ratio configuration parameter 403
grouping effect on access plan 204

H

hash join
described 188
tuning performance of 188
health monitoring configuration parameter 510
health_mon configuration parameter 510

I

I/O
configuration parameters 431
I/O parallelism
managing 282
INCLUDE clause
effect on space required for indexes 23
index re-creation time configuration parameter 470
index scans
accessing data through 177
previous leaf pointers 31
search processes 31
usage 31
indexes
advantages of 294
block index-scan lock mode 80
cluster ratio 183
clustering 23
collecting catalog statistics on 123
data-access methods using 181
defragmentation, online 305
detailed statistics data collected 142
effect of type on next-key locking 86
index re-creation time configuration parameter 470
managing 294, 302
managing for MDC tables 28
managing for standard tables 23
performance tips for 299
planning 296
reorganizing 303
rules for updating statistics manually 155
scans 31
structure 31
type-2 described 302
when to create 296
wizards to help design 242
indexrec configuration parameter 470
initial number of agents in pool configuration parameter 450
initial number of fenced processes configuration parameter 453
inserting data
process for 34
when table clustered on index 34
instance memory configuration parameter 421

instance_memory configuration parameter 421
intra_parallel configuration parameter 506
intra-partition parallelism
optimization strategies for 206
IS (intent share) mode 61
isolation levels
effect on performance 52
locks for concurrency control 59
specifying 57
statement-level 57

J

Java Development Kit installation path (DAS) configuration parameter 535
Java Development Kit installation path configuration parameter 519
java_heap_sz configuration parameter 426
jdk_path configuration parameter 519
jdk_path DAS configuration parameter 535
joins
broadcast inner-table 198
broadcast outer-table 198
collocated 198
db2expln information displayed for 620
described 187
directed inner-table 198
directed inner-table and outer-table 198
directed outer-table 198
eliminating redundancy 168
hash, described 188
in partitioned database 198
merge, described 188
methods listed 188
nested-loop, described 188
optimizer strategies for optimal 191
shared aggregation 168
subquery transformation by optimizer 168
table-queue strategy in partitioned databases 196

K

keep fenced process configuration parameter 450
keepfenced configuration parameter 450

L

large objects (LOBs)
caching behavior for 307
list prefetching 278
LOCK TABLE statement
to minimize lock escalations 70
locking
maximum percent of lock list before escalation 428
maximum storage for lock list 397
time interval for checking deadlock configuration parameter 427
tuning for 68
locklist configuration parameter
description 397
effect on query optimization 163
locks
block index-scan modes 80
configuration parameters 427
deadlocks 14
effect of application type 84
effect of data-access plan 85
escalation
correcting 70
defined 59
preventing 70
exclusive (X) mode 61
intent exclusive (IX) mode 61
intent none (IN) mode 61
intent share (IS) mode 61
lock modes for table and RID index scans for MDC tables 77
modes and access paths for standard tables 74
next-key locking 86
performance factors 63
share (S) mode 61
share with intent exclusive (SIX) mode 61
superexclusive (Z) mode 61
type-compatibility tables 72
types 61
update (U) mode 61
LOCKSIZE clause 59
locktimeout configuration parameter 430
log buffer 33
log file space
required for data redistribution 352
log records
user exit enable configuration parameter 468
log_retain_status configuration parameter 488
LOGBUFSZ configuration parameter 395
LOGFILSIZ configuration parameter 454
logging
circular, defined 33
retain log records, defined 33
loghead configuration parameter 464
logical nodes 36
logical partitions
multiple 36
logpath configuration parameter 463
LOGPRIMARY configuration parameter 456
logretain configuration parameter 467
logs
configuration parameters affecting log activity 464
configuration parameters affecting log files 454
created by governor tool 333
first active log file configuration parameter 464
location of log files configuration parameter 463
log buffer size configuration parameter 395
log retain enable configuration parameter 467
log retain status indicator configuration parameter 488
mirror log path 461
newlogpath configuration parameter 459
number of primary log files configuration parameter 456
number of secondary log files configuration parameter 458
overflow log path 462
recovery range and soft checkpoint interval configuration parameter 465
size of log files configuration parameter 454
LOGSECOND configuration parameter
description 458
long fields
caching behavior for 307

M

map pages
  extent 20
  space 20
materialized query tables
  automatic summary tables 210
  replicated, in partitioned databases 194
max_connretries 503
max_coordagents configuration parameter 446
max_logicagents configuration parameter 448
max_querydegree configuration parameter 505
max_time_diff configuration parameter 504
maxagents configuration parameter 444
  effect on memory use 251
  for memory management 44
maxappls configuration parameter 438
  effect on memory use 251
  for memory management 44
maxcagents configuration parameter 445
maxcoordagents configuration parameter 251
maxfilop configuration parameter 441
maximum database files open per application configuration parameter 441
maximum Java interpreter heap size configuration parameter 426
maximum number of active applications configuration parameter 438
maximum number of agents configuration parameter 444
maximum number of concurrent agents configuration parameter 445
maximum number of concurrently active databases configuration parameter 514
maximum number of coordinating agents configuration parameter 446
maximum number of fenced processes configuration parameter 452
maximum percent of lock list before escalation configuration parameter 428
maximum query degree of parallelism configuration parameter
  description 505
  effect on query optimization 163
maximum size of application group memory set configuration parameter 402
maximum storage for lock list configuration parameter 397
maximum time difference among nodes configuration parameter 504
maxlocks configuration parameter 428
maxtotfilop configuration parameter 442
memory
  agent communication 416
  agent private 405
  applheapsz configuration parameter 410
  application communication 416
  application shared 401
  aslheapsz configuration parameter 417
  buffer-pool allocation at startup 271
  database manager instance 421
  database shared 391
  dbheap configuration parameter 392
  global, components of 258
  instance memory configuration parameter 421
  organization of use 251
  package cache size configuration parameter 399
  sort heap size configuration parameter 405
  sort heap threshold configuration parameter 406
  statement heap size configuration parameter 409
  tuning parameters that affect 261
  when allocated 251
memory model
  database-manager shared memory 254
  described 44
memory requirements
  FCM buffer pool 256
merge join 188
min_dec_div_3 configuration parameter 419
min_priv_mem configuration parameter 414
mincommit configuration parameter 464
MINPCTUSED clause
  for online index defragmentation 23
mirror log path configuration parameter 461
mirrorlogpath configuration parameter 461
modeling application performance
  using catalog statistics 149
  using manually adjusted catalog statistics 147
mon_heap_sz configuration parameter 422
monitor switches
  updating 316
monitoring
  how to 316
multidimensional clustering (MDC)
  management of tables and indexes 28
  optimization strategies for 209
multipage_alloc configuration parameter 488
  effect on memory 19
  setting for SMS table spaces 19
multisite update 51
  configuration parameters 476

N

nested-loop join 188
NetBIOS
  workstation name configuration parameter 496
newlogpath configuration parameter 459
next-key locks
  converting index to minimize 303
  effect of index type 86
  in type-2 indexes 302
nname configuration parameter 496
node 51
node connection retries configuration parameter 503
nodes
  connection elapse time 501

nodes *(continued)*
coordinating agents, maximum 446
maximum time difference among 504
message buffers, number 502
nodetype configuration parameter 518
notifylevel configuration parameter 509
num_db_backups configuration parameter 473
num_estore_segs configuration parameter
description 437
for memory management 44
num_freqvalues configuration parameter 493
num_initfenced configuration parameter 453
num_iocleaners configuration parameter 432
num_ioservers configuration parameter 434
num_poolagents configuration parameter 448
num_quantiles configuration parameter 494
number of commits to group configuration parameter 464
number of database backups configuration parameter 473
numdb configuration parameter 514
effect on memory use 251
for memory management 44
numinitagents configuration parameter 450
numsegs configuration parameter 436
NW (next key weak exclusive) mode 61

O

online
help, accessing 656
operations
merged or moved by optimizer 166
optimization
intra-partition parallelism 206
strategies for MDC tables 209
optimization classes
choosing 88
listed and described 90
optimization classes *(continued)*
setting 93
OPTIMIZE FOR clause
in query tuning 95
optimizer
access plan
effect of sorting and grouping 204
for column correlation 174
index access methods 181
using index 177
distribution statistics, use of 136
joins
described 187
in partitioned database 198
strategies for optimal 191
query rewriting methods 166
ordering DB2 books 656
overflow records
in standard tables 23
performance effect 288
overflowlogpath configuration parameter 462
overhead
row blocking to reduce 99

P

page cleaners
tuning number of 267
pages, data 23
parallel
configuration parameters 501
parallel processing, information displayed by db2expln output 626
parallelism
effect of
dft_degree configuration parameter 107
intra_parallel configuration parameter 107
max_querydegree configuration parameter 107
I/O
managing 282
server configuration for 279
intra-partition
optimization strategies 206
non-SMP environments 107
setting degree of 107
partition groups, effect on query optimization 111
partitioned database
configuration parameters 501
partitioned databases
data redistribution, error recovery 353
decorrelation of a query 171
errors when adding nodes 344
join methods in 198
join strategies in 196
replicated materialized query tables in 194
partitions
adding
to a running system 339
to a stopped system 342
to NT system 341
dropping 346
pckcachesz configuration parameter 399
PCTFREE clause
to retain space for clustering 23
performance
adjusting optimization class 93
db2batch benchmarking tool 358
developing improvement process 5
disk-storage factors 15
elements of 3
federated database systems 213
limits to tuning 7
tuning 4
performance monitor
using 316
performance-tuning process
quick-start tips 7
user input for 6
point-in-time monitoring 316
pool size for agents, controlling 448
precompiling
isolation level 57
predicates
applying 171
characteristics 184
implied
added by optimizer 173
translated by optimizer 166
prefetching
block-based buffer pools 277
description 274
I/O server configuration for 279
intra-parallel performance 274
list sequential 278
parallel I/O 280
sequential 275
printed books, ordering 656

priv_mem_thresh configuration parameter 415
process model
  for SQL compiler 159
  for updates 35
  overview 36
pushdown analysis
  for federated database queries 213

Q

quantile distribution statistics 133
queries
  tuning
    guidelines 101
    restricting select statements 95
    SELECT statements 101
query optimization
  configuration parameters affecting 163
  effect of partition groups 111
query_heap_sz configuration parameter 411

R

rec_his_retentn configuration parameter 473
record identifier (RID), in standard tables 23
recovery
  configuration parameters 469
recovery history retention period configuration parameter 473
recovery range and soft checkpoint interval configuration parameter 465
registry variables
  DB2_ANTIJOIN 556
  DB2_APM_PERFORMANCE 562
  DB2_AVOID_PREFETCH 562
  DB2_AWE 562
  DB2_BINSORT 562
  DB2_CORRELATED_PREDICATES 556
  DB2_DARI_LOOKUP_ALL 562
  DB2_ENABLE_BUFPD 562
  DB2_EXTENDED_OPTIMIZATION 562
  DB2_HASH_JOIN 556
  DB2_INLIST_TO_NLJN 556
  DB2_LIKE_VARCHAR 556
  DB2_MINIMIZE_LIST_PREFETCH 556
  DB2_MMAP_READ 562
  DB2_MMAP_WRITE 562
  DB2_NEW_CORR_SQ_FF 556
  DB2_OVERRIDE_BPF 562
registry variables *(continued)*
  DB2_PINNED_BP 562
  DB2_PRED_FACTORIZE 556
  DB2_REDUCED_OPTIMIZATION 556
  DB2_SELECTIVITY 556
  DB2_SORT_AFTER_TQ 562
  DB2_STPROC_LOOKUP_FIRST 562
  DB2ACCOUNT 542
  DB2ADMINSERVER 571
  DB2ATLDPORTS 554
  DB2ATLDPWFILE 554
  DB2BIDI 542
  DB2BPVARS 562
  DB2BQTIME 553
  DB2BQTRY 553
  DB2CHECKCLIENTINTERVAL 548
  DB2CHGPWDESE 554
  DB2CHKPTR 562
  DB2CODEPAGE 542
  DB2COMM 548
  DB2CONNECTIN_APP_PROCESS 546
  DB2DBDFT 542
  DB2DBMSADDR 542
  DB2DEFPREP 571
  DB2DISABLE_FLUSH_LOG 542
  DB2DISCOVERYTIME 542
  DB2DJ_COMM 571
  DB2DMNBCKCTLR 571
  DB2DOMAINLIST 546
  DB2ENABLE_LDAP 571
  DB2ENVLIST 546
  DB2FALLBACK 571
  DB2FORCE_FCM_BP 554
  DB2FORCE_NLS_CACHE 548
  DB2GRP_LOOKUP 571
  DB2INCLUDE 542
  DB2INSTANCE 546
  DB2INSTDEF 542
  DB2INSTOWNER 542
  DB2INSTPROF 546
  DB2IQTIME 553
  DB2LDAPBASEDN 571
  DB2LDAPCACHE 571
  DB2LDAPCLIENT_PROVIDER 571
  DB2LDAPHOST 571
  DB2LDAPSEARCH_SCOPE 571
  DB2LIBPATH 546
  DB2LIC_STAT_SIZE 542
  DB2LOADREC 571
  DB2LOCALE 542
  DB2LOCKTO_RB 571
  DB2MAXFSCRSEARCH 562
  DB2MEMDISCLAIM 562
  DB2MEMMAXFREE 562
  DB2NBADAPTERS 548
registry variables *(continued)*
  DB2NBBRECVNCBS 548
  DB2NBCHECKUPTIME 548
  DB2NBDISCOVERRCVBUFS 542
  DB2NBINTRLISTENS 548
  DB2NBRECVBUFFSIZE 548
  DB2NBRESOURCES 548
  DB2NBSENDNCBS 548
  DB2NBSESSIONS 548
  DB2NBXTRANCBS 548
  DB2NETREQ 548
  DB2NEWLOGPATH2 571
  DB2NODE 546
  DB2NOEXITLIST 571
  DB2NTMEMSIZE 562
  DB2NTNOCACHE 562
  DB2NTPRICLASS 562
  DB2NTWORKSET 562
  DB2NUM_FAILOVER_NODES 554
  DB2OPTIONS 542
  DB2PARALLEL_IO 546
  DB2PATH 546
  DB2PORTRANCE 554
  DB2PRIORITIES 562
  DB2REMOTEPREG 571
  DB2RETRY 548
  DB2RETRYTIME 548
  DB2ROUTINEDEBUG 571
  DB2RQTIME 553
  DB2SERVICETPINSTANCE 548
  DB2SLOGON 542
  DB2SORCVBUF 571
  DB2SORT 571
  DB2SOSNDBUF 548
  DB2SYSPLEXSERVER 548
  DB2SYSTEM 571
  DB2TCPCONNMGRS 548
  DB2TERRITORY 542
  DB2TIMEOUT 542
  DB2TRACEFLUSH 542
  DB2TRACENAME 542
  DB2TRACEON 542
  DB2TRCSYSERR 542
  DB2USE_PAGE_CONTAINER_TAG 546
  DB2VENDOR_INI 571
  DB2VI_DEVICE 548
  DB2VI_ENABLE 548
  DB2VI_VIPL 548
  DB2XBSA_LIBRARY 571
  DB2YIELD 542
  DLFMBACKUP_DIR_NAME 569
  DLFMBACKUP_LOCAL_MP 569
  DLFMBACKUP_TARGET 569
  DLFMBACKUP_TARGET_LIBRARY 569
  DLFMENABLE_STPROC 569

registry variables *(continued)*
DLFMFS_ENVIRONMENT 569
DLFMGC_MODE 569
DLFMINSTALL_PATH 569
DLFMLOG_LEVEL 569
DLFMPORT 569
DLFMTSM_MGMTCLASS 569
overview 541
release configuration parameter 481
remote data services node name configuration parameter 496
REORG INDEXES command 303
REORG TABLE command
choosing reorg method 291
classic, in off-line mode 291
in-place, in on-line mode 291
REORGANIZE TABLE command
indexes and tables 303
reorganizing
tables
determining when to 288
restore_pending configuration parameter 488
resync_interval configuration parameter 477
REXX language
isolation level, specifying 57
roll-forward recovery
definition 33
rollforward utility
roll forward pending indicator 487
rollfwd_pending configuration parameter 487
row blocking
specifying 99
rows
lock types 61
rqrioblk configuration parameter 420

S

SARGable
defined 184
sched_enable configuration parameter 532
sched_userid configuration parameter 536
search discovery communications protocols configuration parameter 499
SELECT statement
eliminating DISTINCT clauses 171
prioritizing output for 95
seqdetect configuration parameter 434
sequential prefetching
described 275
SET CURRENT QUERY OPTIMIZATION statement 93
shadow paging, long objects 33
sheapthres configuration parameter 406
sheapthres_shr configuration parameter 408
SIX (share with intent exclusive) mode 61
smtp_server configuration parameter 534
snapshots
point-in-time monitoring 316
softmax configuration parameter 465
sortheap configuration parameter
description 405
effect on query optimization 163
sorting
effect on access plan 204
managing 284
sort heap size configuration parameter 405
sort heap threshold configuration parameter 406
sort heap threshold for shared sorts 408
space map pages (SMP), DMS table spaces 20
spm_log_file_sz configuration parameter 479
spm_log_path configuration parameter 478
spm_max_resync configuration parameter 479
spm_name configuration parameter 478
SQL compiler
process description 159
SQL statements
benchmarking 356
statement heap size configuration parameter 409
SQLDBCON configuration file 369
start and stop timeout configuration parameter 504
start_stop_time configuration parameter 504
stat_heap_sz configuration parameter 410
statement heap size configuration parameter 409
statement-level isolation, specifying 57
static SQL
setting optimization class 93
stmtheap configuration parameter 409
stmtheap configuration parameter, effect on query optimization 163
stored procedures
configuration parameters 450
how used 106
subqueries
correlated
how rewritten 171
summary tables
See materialized query tables. 210
svcename configuration parameter 497
sysadm_group configuration parameter 521
sysctrl_group configuration parameter 522
sysmaint_group configuration parameter 523
system managed space (SMS)
described 19
system management
configuration parameters 512

T

table spaces
DMS 20
effect on query optimization 111
lock types 61
overhead 111
TRANSFERRATE, setting 111
types
SMS or DMS 19
tables
access
information displayed by db2expln 611
paths 74
lock modes
for RID and table scans of MDC tables 77
for standard tables 74
lock types 61
multidimensional clustering 28
queues, for join strategies in partitioned databases 196

tables *(continued)*
reorganization
classic, in off-line mode 287
determining need for 288
in-place, in on-line mode 287
reducing need for 287
reorganizing 291
standard
managing 23
TCP/IP service name configuration
parameter 497
temporary tables
use information, db2expln 617
territory configuration
parameter 482
threads
description 36
time
deadlock configuration
parameter, interval for checking 427
difference among nodes, maximum 504
Tivoli Storage Manager (TSM)
configuration parameters 469
tm_database configuration
parameter 476
toolscat_db configuration
parameter 533
toolscat_inst configuration
parameter 532
toolscat_schema configuration
parameter 533
tp_mon_name configuration
parameter 516
tpname configuration
parameter 498
track modified pages enable
configuration parameter 474
trackmod configuration
parameter 474
triggers
Explain tables 577
troubleshooting
DB2 documentation search 664
online information 666
trust_allclnts configuration
parameter 527
trust_clntauth configuration
parameter 528
tsm_mgmtclass configuration
parameter 474
tsm_nodename configuration
parameter 475
tsm_owner configuration
parameter 476
tsm_password configuration
parameter 475
tuning
configuration parameters 371
tutorials 668
type-2 indexes
advantages of 302
described 31
next-key locking in 86

U

use_sna_auth configuration
parameter 525
user defined functions
configuration parameters 450
user exit enable configuration
parameter 468
user exit status indicator
configuration parameter 488
user_exit_status configuration
parameter 488
user-defined functions (UDFs)
entering statistics for 146
userexit configuration
parameter 468
util_heap_sz configuration
parameter 396

V

views
merging by optimizer 168
predicate pushdown by optimizer 171

W

W (Weak Exclusive) mode 61
WHERE clause
predicate terminology definitions 184
Windows
adding partitions 341
workloads
Workload Performance wizard 242

X

X (Exclusive) mode 61

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

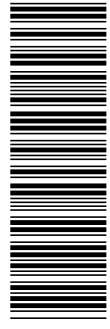
If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Part Number: CT17YNA

Printed in U.S.A.



(1P) P/N: CT17YNA

SC09-4821-00

Spine information:

IBM® DB2 Universal Database™ Administration Guide: Performance Version 8